Order Number 9219387

Development of a cognitive diagnosis system for intelligent tutoring: An application to transportation problem-solving

Shin, Dong-Ik, Ph.D.

The University of Nebraska - Lincoln, 1992

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

DEVELOPMENT OF A COGNITIVE DIAGNOSIS SYSTEM FOR INTELLIGENT TUTORING: AN APPLICATION TO TRANSPORTATION PROBLEM SOLVING

by

Dong-Ik Shin

A DISSERTATION

Presented to the Faculty of

The Graduate College in the University of Nebraska

In Partial Fulfillment of Requirements

For Degree of Doctor of Philosophy

Major: Interdepartmental Area of Business

Under the Supervision of Professor Sang M. Lee

Lincoln, Nebraska

December, 1991

DISSERTATION TITLE

Development of A Cognitive Diagnosis System For Intelligent Tutoring :

An Application To Transportation Problem Solving

BY

Dong-Ik Shin

SUPERVISORY COMMITTEE:

	APPROVED	DATE
Signature	Sang M. Lee	12/17/91
Typed Name	Sang M. Lee	
Signature	Lester A. Digman	12/17/91
Typed Name	Lester A. Digman	
Signature	James F. Brown	12/17/91
Typed Name	James F. Brown	
Signature	Susan Wiedenbeck	12/17/91
Typed Name	Susan Wiedenbeck	
Signature		
Typed Name		
Signature		
Typed Name		

GRADUATE COLLEGE
UNIVERSITY OF NEBRASKA

DEVELOPMENT OF A COGNITIVE DIAGNOSIS SYSTEM FOR INTELLIGENT TUTORING : AN APPLICATION TO TRANSPORTATION PROBLEM SOLVING

Dong-Ik Shin, Ph.D.
University of Nebraska, 1991

Adviser : Sang M. Lee

The purpose of this study is to design and develop a cognitive diagnosis system for intelligent tutoring. The cognitive diagnosis system proposed in this dissertation is called PSCD (Production System for Cognitive Diagnosis). PSCD is intended for intelligent tutoring with possible relevance to decision aiding.

This dissertation is based on two assumptions. First, it is assumed that perspectives and methodologies of cognitive science provide a starting point for integrating decision-making studies and cognitive studies (Simon, 1979). Second it is assumed that integrating theories and building an integrated theory or theories into a cognitive architecture provide opportunities to realize practical applications (Newell, 1990). Basing on these two assumptions, the study attempts to integrate three theories of human information processing and to build the integrated theory into PSCD.

Three theories considered in this study are heuristic search, production systems architecture, and constructivism. These three theories act as constraints in setting the design specification of PSCD in the huge space of alternative designs. PSCD that are developed henceforth is applied to a

counting task for initial validation, and to a transportation problem solving task which is the main task of this dissertation.

ACKNOWLEDGEMENT

In the process of this dissertation research, I have benefited from many people. First , I would like to thank my dissertation adviser, Dr. Sang M. Lee, for his intellectual and moral support, and the members of my dissertation committee for their helpful suggestions and support. My sincere thanks also go to my friends and their families for making the times so endurable.

Finally, I would like to thank my parents and family for their encouragement and financial support which gave me the strength to complete this work.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1 INTRODUCTION

The purpose of this study is to design and develop a computational model of cognitive diagnosis for intelligent tutoring. The cognitive diagnosis system proposed in this dissertation is called Production System for Cognitive Diagnosis (PSCD). PSCD is intended for intelligent tutoring with possible extension to decision aiding. Cognitive diagnosis refers to the process of inferring a person's cognitive state from his performance (Ohlsson, 1986a; VanLehn, 1988). Cognitive diagnosis is a central problem in developing an intelligent tutoring system (ITS) which is a computer system that helps students learn. In addition, cognitive diagnosis is an essential capability of a knowledge communication system. Knowledge communication is the idea that human-computer interactions can be viewed as causing or supporting the acquisition of one's knowledge by the other, via a restricted set of communication operation (Wenger, 1987). Given this definition, the idea of knowledge communication is applicable without suspicion to ITSs, also appropriate to expert systems and even to any intelligent computer systems that are intended for human uses.

Cognitive diagnosis is a difficult problem because a person's cognitive states are not observable and must be inferred from observable behaviorai data. In addition, there is a methodological problem in determining a unique representation and process, used to describe cognitive states, that might have produced observed behavioral. It is generally known that the same behavioral data can be reproduced by a number of different cognitive representations and processes (Anderson, 1990). The problem is called the nonidentifiability problem.

The nonidentifiability problem is viewed so serious and intractable that Anderson (1990), for example, proposed to avoid inferring cognitive states from

behavioral data but to determine cognitive states on the basis of the normative theory of optimization. The Anderson's (1990) approach to the nonidentifiability problem, as Newell (1990) pointed out, arises from a narrow focus on behavioral data in identifying a unique representation and process. Another way of overcoming the nonidentifiability problem is to integrate large and diverse collections of knowledge that would help pin down to a unique representation and process of cognitive states. A repository of such large and diverse knowledge is called a unified theory (Newell, 1990).

A unified theory is argued to result in great increases in identifiability. The unified theory approach calls for brining more constraints in the design of a system and approximating real situations as close as possible. Through the process of developing a unified theory, implementing it in a system, and comparing behaviors of a system to real behavioral data, there is more opportunity to discover impacts and roles of each constraint and to explore validity of alternative design specifications. The unified theory approach reduces the degrees of freedom that allow so many models to coexist with the same data. Such a way of eliminating alternative models is termed as the sufficiency criterion (Newell, 1973a).

The approach taken in this study is the unified theory approach. The unified theory approach merits attention because it provides a way of overcoming the nonidentifiability problem. In addition, it is the unified theory approach that opens the way for practical application, for example the cognitive diagnosis system of this study. This is the case because the more unified a theory, the more approximate a system to real situations, and the more a system practically useful.

This cognitive diagnosis system proposed in this study can be used as a component in decision aids. Developing decision aids that would help decision makers make better decisions has become the central concern in decision-making studies. In this introductory chapter, the perspective of this study is outlined and discussed in relation to decision-making studies (Section 1.1) and to various kinds of decision aids (Section 1.2). Section 1.3 shows the overview of this study.

1.1 Decision-Making

Decision-making is a subject which has attracted the attention of scholars of a number of different disciplines (Ungson and Braunstein, 1982; Janis and Mann, 1977; Simon, 1960). Economists, equipped with a theory of rational choice, have studied how human decision-making causes, and is caused by, economic activities (Demski, 1972; Kreps, 1988; Schoemaker, 1991); psychologists, especially behavior decision psychologists, have described and explained the point of departure of human performance from the normative theory (Simon, 1955, 1960, 1976; Hogarth, 1980; Wright, 1985; Hogarth and Reder, 1987; Hogarth, 1990); decision analysts have developed tools intended to help decision makers do what economists often assume they are already doing (Lee, 1972; Raiffa, 1974; Keeney and Raiffa, 1976; Winterfeldt and Edwards, 1986; Lee et al, 1990); and artificial intelligence researchers have tried to develop computer-implemented tools for aiding decisions or for learning how to make decisions (Buchanan and Shortliffe, 1984; Clancey, 1986; Anderson et al., 1985).

Business researchers, drawing concepts and methodologies from a number of different disciplines, have studied individual and organizational

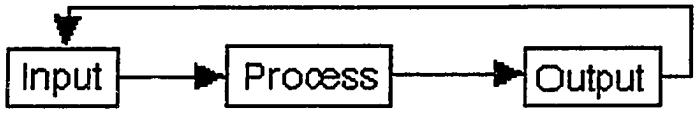
decision-making in a number of different business tasks (Argyris, 1977; Libby, 1981; Bailey, Jr., 1987; Bhaskar and Dillard, 1977). Especially, the behavioral sciences have exerted significant influence on research in business (Ungson and Braunstein, 1982), and business research that uses concepts and methodologies of the behavioral sciences is termed *behavioral research* (Caplan, 1988). The term "behavioral sciences" is usually viewed by business researchers as encompassing psychology, organization theory, and sociology. These three disciplines deal with issues at different levels: psychology deals with issues at the individual level; organization theory at the organization level; and sociology at the society level.

Psychology, especially behavioral psychology, has been extensively applied to a number of business problems (Libby, 1981; Ko and Mock, 1988; Shields, 1988). In the tradition of behavioral psychology, much behavioral research has represented human decision-making in statistical models such as regression (Hursch et al., 1964; Slovic et al., 1977; Libby, 1975), ANOVA (Hoffman et al., 1968; Ashton, 1974), and multidimensional scaling (Green and Rao, 1972; Libby, 1979), or in variants of statistical decision theory such as heuristics (Tversky and Kahneman, 1974; Joyce and Biddle, 1981; Waller and Felix, 1987). Although this body of research has resulted in useful insights and an extensive amount of empirical data, criticisms have also been raised. Many studies that used linear models to represent HIP showed very high levels of predictive accuracy. The very high predictive power of such linear models, however, is argued to be largely an artifact of the mathematics, because linear models were found to be very robust even when assumptions were violated (Dawes and Corrigan, 1974). Research in cognitive heuristics and biases also was subject to such criticisms as lack of attempts to understand decision

processes and to build decision aids which would improve decision-making (Shanteau, 1989).

Statistical and heuristics/biases studies have given rise to many different types of theories. Underlying these decision-making studies is the perspective of information processing psychology. Information processing psychology views a decision maker as an information processing system which takes input from the environment, processes input, and produces output (Figure 1.1). Produced output can be further used as input to an information processing system.

Figure 1.1. An informations processing system



Viewing a decision maker as an information processing allows to characterize different perspectives of decision-making studies, because in this view models of decision maker are emphasized. For example, the perspective of economists is a rational man, statistical decision-making studies view a decision maker as an intuitive statistician, and the view of behavioral decision theorists is a fallible man. The perspective taken in this study is a constructive mind, which will be explained in detail Chapter 4. The hypothesized perspective of an information processing system determines the frame of reference of the study (Clancey, 1991). Each perspective and associated methodologies shape the degree and extent of models of decision maker. In fact, it appears that decision-making studies have progressed to develop more

descriptively valid and powerful models of decision maker. In other words, decision-making researchers have attempted to develop models of decision maker that are powerful enough to reflect many observed aspects of decision-making.

Thus, this study proposes two requirements for a theory of decision-making. First, a theory of decision-making must have a model of a decision maker. This model is called cognitive model in cognitive science, and often called user model in human-computer interaction literature or student model in intelligent tutoring system literature. Studies of decision-making can be viewed progressing to more powerful representations or models of decision maker. Second, a theory of decision-making must be able to predict and explain human decision behavior with a model of a decision maker. This includes not only correct performance but also faulty performance. Only then can a theory of decision-making be realized in a decision aid that is able to provide "true" support to decision maker, and prove its usefulness. In fact, a final test for a theory of decision-making is its utility for aiding and improving decisions.

Reviewing decision-making studies led to two general conclusions: (a) humans tend to err and (b) even this tendency, humans can successfully perform difficult tasks, given time, help, and tools (Libby, 1981; Reason, 1990; Winterfeldt and Edwards, 1986). These observations led to increased interest in understanding human errors, and in designing and developing decision aids which may help humans avoid such errors.

There is a variety of decision aids which rely on different theories and methodologies. This study employs theories and methodologies of cognitive science to propose a model of cognitive diagnosis. This study assumes that abilities of both decision-making and problem-solving originate from one mind,

though there is a difference in the type of tasks being addressed. Decision-making typically involves ill-structured problems whereas problem-solving usually concerns well-structured problems. It may be the case that there are some invariants that do not differ in both cognitive processes. Those invariants constitute a functional architecture of mind. After all, as Newell (1980, 1990) observed, it is one mind that solves a problem, makes a decision, and that produces all aspects of intelligent behavior.

This study addresses the problem of cognitive diagnosis, often called user modeling or student modeling. Cognitive diagnosis can be used for several purposes: determining if the user's performance is correct by comparing an inferred model to an expert model; explaining why the performance is wrong; predicting his or her performance in future tasks. Such abilities enable many applications to be possible: preventing a human error that can make a catastrophic disaster in a dangerous situation, for example nuclear plant failure; enhancing explanation capability of expert systems, automating protocol analysis; and training novices or students. The intended use of PSCD in this study is for intelligent tutoring.

1.2 Decision Aids

Designing decision aids that improve decision-making has been the central concern in Management Information System (MIS), and especially in the subfield of MIS, known as Decision Support System (DSS). MIS is a computer-based system for collecting, storing, retrieving, and processing information that is used, or desired, by managers in the performance of their duties (Ein-Dor and Segev, 1977; Davis and Olson, 1985). MIS typically involves in organizing information so that a decision maker may easily assess and understand

information necessary for performing his or her duties. Such systems have in general proven to be most useful, but the idea behind MIS has a serious limitation. MIS mostly focuses on organizing information, and incorporates very few decision-aiding ideas.

A more powerful system may provide decision models that can aid decisions, as well as easy access to organized information. A class of systems, called DSS, places more emphasis on "support" than does MIS. DSS supports decision makers in their attempts to solve semistructured problems and provides interactive means to test alternative solutions for their consequences (Gorry and Scott Morton, 1971; Keen and Scott Morton, 1978; Alter, 1980). DSS has a more powerful processing capability than MIS in general, but its processing capability is commonly limited to answer "what-if..." questions.

Recent advances in artificial intelligence (AI) have introduced a new way of building the most interesting kinds of decision aids, expert systems. There have been many studies trying to apply AI methodologies to difficult business problems (Paradice and Courtney, 1989; O'Leary, 1987; Vasarhelyi, 1987; Peters et al., 1989; Blanning, 1990; Pau et al., 1989). An expert system is a computer program that exploits the judgments made earlier by an expert in the task at hand. These judgments, which are represented in a computer with the use of a computational language, can provide both expertise and procedural advice that a decision maker may find helpful.

The process of building an expert system is often called knowledge engineering (Waterman, 1986). Knowledge engineering involves extracting from the human experts their procedures, strategies, and rules of thumb for problem solving, and building this knowledge into the expert system. Knowledge engineering extracts humanlike knowledge from experts, but is not

concerned with cognitive fidelity of various ways of using such knowledge. In other words, the knowledge engineering methodology deploys humanlike knowledge in nonhuman ways. One pitfall of using such a methodology is the potential absence of knowledge communication in resulting systems (Anderson, 1988).

Knowledge communication, in fact, is the goal of any computer system intended for human uses. When a computer system lacks cognitive fidelity, a computer system may not understand what a user intends to communicate to it, and a user also may have difficulties in understanding what a computer tries to say. It may be the case that true knowledge communication is possible only when both sides have a model of each other (Anderson, 1988). One requirement for a computer system, then, is to keep a cognitive model of the user, so that it can interpret the user's actions with a cognitive model. This approach is called the *cognitive modeling* or cognitive simulation approach, which is at the heart of cognitive science.

Cognitive modeling, often called computational modeling because cognitive modeling typically presumes the use of a computer for modeling, is more active in ITS studies than in expert systems studies. This is primarily because of the necessity of maintaining a student model in ITS. The cognitive modeling approach has also been advocated as a promising approach for studying human knowledge (Anderson, 1987). The best way to study human knowledge, as Anderson (1987) argued, is to look for differential learning outcomes in pedagogical experiments that manipulate instructional experience, and therefore the ITS paradigm provides a particularly fruitful way to implement such experiments.

The cognitive modeling approach has also been proposed as a reasonable methodology for acquiring a working expert system (Anderson, 1988; Slatter, 1987). Although the constraint of being true to human behavior has been more of a burden than a stimulus, it has also been argued that, by closely mapping between human behavior and cognitive models, it is possible to augment psychological theory and computer science technology as well (VanLehn, 1991).

This study uses the cognitive modeling methodology in its attempt to develop a cognitive diagnosis system. The cognitive modeling methodology is advocated in this study because the methodology allows to build more descriptively valid and also powerful representations of users. The methodology used in this study , however, is different from the traditional cognitive modeling methodology which typically relies on various process tracing methods in order to construct a computational model. In traditional cognitive modeling studies, theories of human information process are constructed typically by analyzing verbal protocols (e.g., Newell and Simon, 1972; Klahr and Wallace, 1976). In this case, theory construction is bottom-up, i.e., from data to theories. In contrast, this study takes the perspective of unified theory, in which theory construction is top-down. In the unified theory approach, theories are embedded into a system and behaviors of a system are compared to data. This study attempts to integrate three theories of HIP and build them into a system, PSCD. These three theories are heuristic search (Newell and Simon, 1976), production system architecture (Klar et al., 1987), and constructivism (Resnick, 1982; Payne and Squibb, 1990).

Unified theories are implemented in cognitive architectures. A cognitive architecture or functional architecture is a fundamental design specification of

an information processing system (Pylyshin, 1984). A cognitive architecture describes the part of HIP that do not change. The notion of architecture differentiates two parts of HIP, the part that changes rapidly and the other part that changes slowly if at all (VanLehn, 1991). Hence, a cognitive architecture is a partial description of a human information processing system (HIP), but describes the essential aspects of HIP. This study proposes that three theories considered in this study must be the part of a cognitive architecture.

1.3 Overview

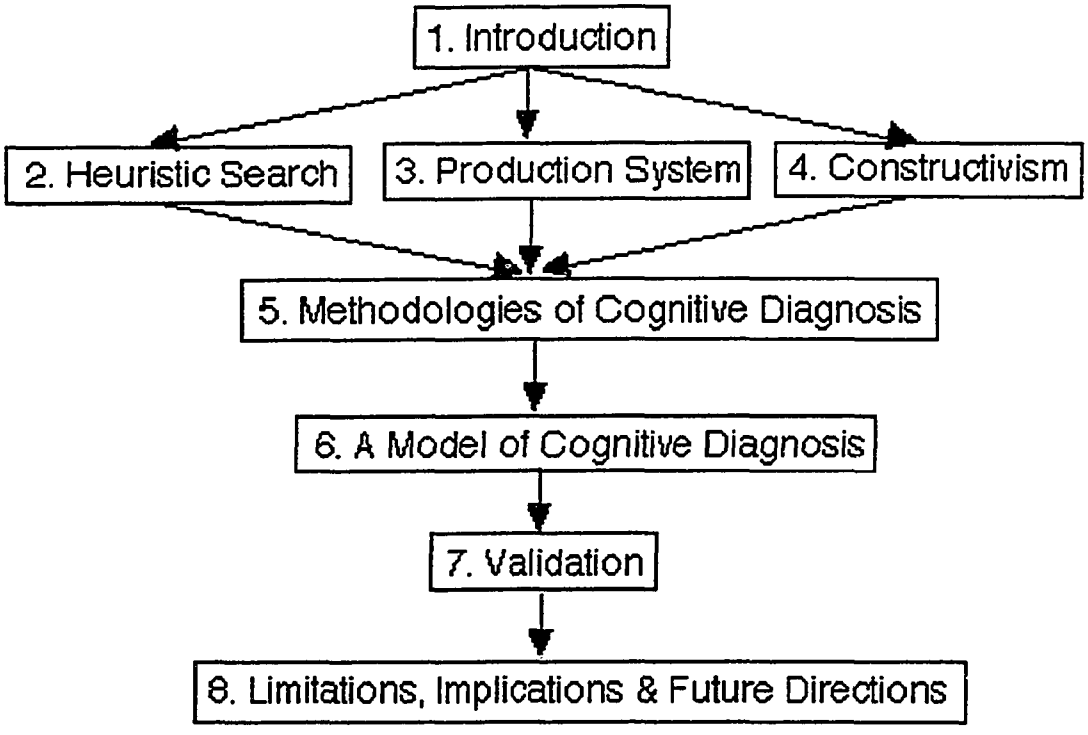
This study integrates three theories of HIP and implement them in PSCD, an architecture for cognitive diagnosis (Figure 1.2). Since these three theories act as constraints in setting the design specification of PSCD in the huge space of alternative designs, they are also called constraints in this study. Two constraints come from major hypotheses that have emerged in cognitive science in the past decade. The first hypothesis is that heuristic search is a fundamental process of HIP (Newell and Simon, 1976; Newell, 1990). The second is that a production system is the architecture of HIP system (Langley, 1983b; Klar et al., 1987; Newell, 1990). In Chapters 2 and 3, these concepts will be defined and explained.

Constructivism is another constraint imbedded in this model. Underlying diverse research done by cognitive, developmental, and educational psychologists is the so-called *constructivist* assumption about how skills are learned (Gelman and Meck, 1986; Resnick, 1984; Brown and VanLehn, 1980; Smith et al., 1989). The constructivist assumption states that knowledge is not directly absorbed but is constructed by each individual. In this view, knowledge is no longer viewed as a reflection of what has been given from the outside; it is

a personal construction from experience and what the individual knows. This construction can be either syntactic (Brown and VanLehn, 1980; Young and O'Shea, 1981) or semantic in nature (Smith et al., 1989; Ohlsson and Rees, 1991). A through review of empirical data and conceptual studies has led to the conclusion that semantic construction is more fundamental in explaining human skill acquisition and performance. This constraint will be discussed in Chapters 4 and 5.

With these constraints in mind, various methodologies of cognitive diagnosis are reviewed in chapter 5. Three constraints and implementing methodologies are finally integrated in Chapter 6. In this chapter, PSCD is formally proposed. Then PSCD is applied to two different tasks for its validation. Chapter 7 discusses some validation issues and presents the results of these applications in Chapter 7. Finally limitations of this study and future directions are discussed in Chapter 8.

Figure 1.2. Overview



Chapter 2 The Heuristic Search Constraint

The heuristic search constraint states that intelligence involves heuristic search (Langley, 1983b; Card et al., 1983). Heuristic search is the major hypothesis in cognitive science. This chapter provides grounds for the use of heuristic search as a constraint in PSCD. Since the concept of heuristic search emerges in cognitive science studies, perspectives of cognitive science are introduced (Section 2.1) and then concepts underlying heuristic search are discussed. The hypothesis of heuristic search is based on three hypotheses that have taken a significant role in many cognitive studies. These hypotheses are the physical symbol system hypothesis (Section 2.2), the problem space hypothesis (Section 2.3), and the bounded rationality hypothesis (Section 2.4), and discussed in the respective section. Finally, Section 2.6 summarized arguments for heuristic search.

2.1 Cognitive Science

The term "cognitive science" has been used in various studies, mostly in artificial intelligence and psychology studies, in a number of different ways (Posner, 1989; Kintsch et al., 1984; Pylyshyn, 1984). Generally, cognitive science is viewed as dealing with the nature of intelligence from the perspective of computation (Posner, 1989). Simon (1989, p. 2) defined cognitive science as "the study of intelligence and its computational processes in humans (and animals), in computers, and in the abstract." This definition shows that the scope of cognitive science studies covers human, computer, and abstract intelligence. Simon provided examples of studies for each category. He regarded experimental and cognitive psychology as an example of the computational study of organismic intelligence because it studies those forms of

intelligence exhibited by people, rats, and pigeons in the laboratory. Computer science, especially the branch of computer science called "artificial intelligence", is provided as an example of the study of intelligence exhibited by machines. Some examples of the study of intelligence in the abstract are formal logic, statistical decision theory, and the theory of maximization of expected utility. This study falls in the category of the study of organismic intelligence because of its concern with human learning and problem solving.

An important word in Simon's definition of cognitive science is "computation." The computational view of cognition is seen as the fundamental assumption in cognitive science (Pylyshyn, 1980, 1984). The view that cognition can be understood as computation is ubiquitous in modern cognitive theorizing, even among those who do not use computer programs to express models of cognitive processes. There has been much discussion whether the human mind can be understood as computation (Simon and Newell, 1972; Pylyshyn, 1984; Anderson, 1983; Newell, 1990; Simon, 1990). Simon and Newell, together and independently, have supplied strong rationales for cognitive science from its inception, and even guided cognitive science research.

The computational view of cognition is based on several assumptions. First, both computers and human cognitions are *artifacts* that adapt to their outer environments in order to satisfy their goals within the limitations of the inner systems (Simon, 1979). The inner system is a organization of the system capable of attaining goals. Since they are artifacts, they both are subject to design. In other words, if the inner system of a system is properly designed, it will be adapted to the outer environment. Second, computer simulation, or often called computational modeling, is a useful methodology to explore

alternative organizational assumptions. Computational modeling is a technique for understanding and predicting the behavior of systems. Simon provided two reasons why computational modeling is useful in finding the laws of behavior governing components. The reasons are that first,, only a few properties abstracted from the complex reality are of interest. The more are abstracted from the detail of a set of phenomena, the easier to simulate. Second, it is not necessary to know entire internal structure but only that part of it that is crucial to the abstraction. Therefore, without identity of the inner systems, simulation is possible because the aspects of interest arise out of the organization of the parts, independent of all but a few properties of the individual components.

From the pioneering works of Simon and Newell, many studies have been conducted in the fields of artificial intelligence and cognitive psychology. Due to extensive interactions and collaborations between these two fields, these two fields collectively merge into a new field known as *cognitive science.*

2.2 The Physical Symbol System Hypothesis

In studying artifacts with the computational modeling methodology, the notions of symbol structures and formal operations are important. In the simplest terms, a symbol is something that stands for something else. This something else is usually called the designation of the symbol. It is the thing that the symbol represents. The designation may be a physical object or a concept, but the symbol itself is physical.

The idea behind symbolic computation is that symbols stand for anything at all. Typically a symbolic program takes as its input one or more symbol structures, representing the initial state of some problem, and returns as its output a symbol structure, representing a terminal state of solution. A symbol

structure is both well-formed in terms of the syntactic rules (which are used to form symbol structures out of symbols in such a way that the resulting structures have a meaning) and has been derived by the application of legal transformations (which turn symbol structures into other symbol structures).

A pioneering study by Newell and Simon (1976), which facilitated the development of the computational view of mind, proposed a *physical symbol system hypothesis*. The physical symbol system hypothesis is stated as follows:

> *Physical Symbol System Hypothesis.* A physical symbol system has the necessary and sufficient means for general intelligent action

A physical symbol system is a machine, located in some environment, with a memory, operators, control, and an input, and over time, it produces an evolving collection of symbol structures. The condition of "necessary" means that any system that exhibits general intelligence will prove to be a physical symbol system. The condition of "sufficient" means that any physical symbol system of sufficient size can be organized further to exhibit general intelligence. From these claims follow two empirical hypotheses: 1. that computers can be programmed to think (sufficient), and 2. that the human brain is a physical symbol system (necessary). These hypotheses are tested by programming computers to perform the same tasks that researchers used to judge how well people are thinking, and then by showing that the processes used by the computer programs are the same as those used by people performing these tasks. Thinking-aloud protocols, records of eye movement, reaction times, and many other kinds of data are used as evidence to make the comparison. Simon (1990) argued that the physical symbol system hypothesis has been tested so

extensively over the past 30 years that it can now be regarded as fully established.

The hypothesis dictates that human cognition can be realizable in a physical symbol system because a computer is a member of a family of artifacts called physical symbol system as human cognition does.

2.3 The Problem Space Hypothesis

Newell (1980, 1990) argued that the problem space representation fundamental to all physical symbol systems, including all categories of cognition such as reasoning (Johnson-Laird, 1983; Johnson-Laird, 1990), problem-solving (Newell and Simon, 1972; Simon, 1983), and decision processes (Slovic et al, 1977; Kahneman et al., 1982). Newell and Simon (1972) derived the concept of a problem space from extensive work in artificial intelligence, and provided a prototypic example of the concept. Since then, many studies characterized as heuristic search have analyzed problem solving tasks in problem space terms. The problem space hypothesis is stated formally as follows (Newell, 1980):

> *Problem Space Hypothesis.* The fundamental organizational unit of all human goal-oriented symbols activity is the problem space.

> *Problem Space.* A problem space consists of a set of symbolic structures (the states of the space) and a set of operators over the space. Each operator takes a state as input and produces a state as output, although there may be other inputs and outputs as well. The operators may be partial (i.e., not defined for all states). Sequences of operators define paths that thread their way through sequences of states.

Problem. A problem in a problem space consists of a set of initial states, a set of goal states, and a set of path constraints. The problem is to find a path through the space that starts at any initial state, passes only along paths that satisfy the path constraints, and ends at any goal state.

Given a problem in a problem space, the only way a system can solve the problem is by searching in the space, working out from the current state by applying operators, adding new states to the stock to be used at new points of search, evaluating whether the result help, etc. Accomplishing this search requires performing repeatedly a fixed set of functions, namely search control. A search system can thus be defined as a problem-solving system consisting of three main components: a database, operators, and control strategy (Barr and Feigenbaum, 1981). A database describes both the current task-domain situation and the goal. The database can have a variety of data structures including arrays, lists, sets of predicate calculus expressions, property list structures, and semantic networks. The second component of a search system is a set of operators used to manipulate the database. The third component of a search system is a control strategy for deciding what to do next—in particular what operator to apply and where to apply it.

A search space represents the structure of a problem in terms of the alternatives available at each possible state of the problem. The basic idea is that from a given state in a problem, all-possible next states can be determined with a small set of rules, called transition operators. For example, in a chess game, the original state is the board position at the beginning of the game. The legal-move generators correspond to the rules for moving each piece. So all of the next states of the game (i.e., the board configurations after each of White's possible first moves) can be generated by applying the move generators to the

original positions of the pieces. Similarly, all the possible states after Black's first response can be generated.

A somewhat straightforward way to find the winning move is to try all the alternative moves, then try all the opponent's responses to these moves, and then try all the possible responses to those, until all the possible continuations of the game have been exhausted and it is clear which was optimal. The problem with this solution is that, for interesting problems like chess, there are far too many possible combinations of moves to try in a reasonable amount of time on a machine of conceivable computational power. This problem, called *combinatorial explosion*, is an important general difficulty for AI systems in all applications.

2.4 The Bounded Rationality Hypothesis

The problem of combinatorial explosion shows the limitation of a physical symbol system in adapting its behavior to the requirements of a given task. This limitation arises because the inner environment of the system places limits on the kinds of information processing which the system is capable of. In the case of computers, adaptiveness is limited by the theorems of Gödel, which prove that every symbol processing system must be incomplete. Far more important than the Gödel limits, Simon (1976) argued, are the limits imposed by the speed and organization of a system's computations, and sizes of its memories. For example, playing a perfect game of chess by using the game-theoretic minimaxing algorithm is one such infeasible computation, for it calls for the examination of more chess positions than there are molecules in the universe.

Similarly, studies of human cognition have revealed that computational capabilities of the human mind are limited because of the limited capacity of short-term memory and the time required to fixate an item. The facts that human

short-term memory can hold only a half dozen chunks that an act of recognition takes nearly a second, and that the simplest human reactions are measured in tens and hundreds of milliseconds instead of microseconds or picoseconds are basic physiological constants that determine what kinds of computations are feasible in a given type of task situation and how rapidly they can be carried out. From these facts, Simon (1955) proposed one of the essential laws of qualitative structure applying to physical symbol systems, including computers and the human brain, namely bounded rationality.

> *Bounded Rationality.* Because of the limits on their computing speeds and power, intelligent systems must use approximate methods to handle most tasks.

2.5 The Heuristic Search Hypothesis

A major way to relax the limits of bounded rationality and to overcome the problem of combinatorial explosion is to store in long-term memory knowledge and strategies that reduce the computational requirements of tasks. In tasks of any complexity, knowledge and strategies do not allow the expert to find an optimal solution, but only to find approximations far better than those available to native (or naive) intelligence. Simon (1990) suggested that it is possible that some common properties, deriving from human bounded rationality, are shared by the approximating procedures people use in many kinds of complex situations.

Knowledge and strategies take a role of limiting the number of alternatives searched at each stage of the look-ahead process to the best possibilities. Newell (1990, pp. 98-100) called this type of search as *knowledge search*, which is "the search in the memory of the system for the knowledge to guide the problem search," and distinguished it from *problem search*, which is

"the search of the problem space." According to Newell, knowledge search happens in the inner loop of problem search, and it searches for knowledge that can be used to evaluate operators and to evaluate their consequences, that is, the resulting new states. Hence, knowledge, often called search control knowledge, can take a role in evaluating operators as well as in evaluating states. Newell further observed that one could always transfer knowledge from the test to the generator, so that new states never were created at all. I.e., there is a trade-off relationship between knowledge for operators' evaluation and knowledge for states' evaluation. Wherever knowledge is put in, either in the generator or in the test, the role of knowledge in problem search is to reduce the problem space, and consequently to increase efficiency (Minton, 1988). Consequently, it is hypothesized that physical symbol systems solve problems by using the processes of heuristic search. More formally, the law is stated as follows:

> *Heuristic Search Hypothesis.* A physical-symbol system exercises its intelligence by heuristic search, that is, by generating and progressively modifying symbol structures with the help of knowledge until it produces a solution structure.

Similar to the distinction between knowledge search and problem search is the distinction between generative and evaluative selectivity (Ohlsson and Rees, 1991). Generative selectivity operates through strategic rules that propose good moves. Strategic rules improve the efficiency of search by focusing attention on the most promising *actions* in each state. Evaluative selectivity operates through evaluation functions that measure the promise of a *state*. Evaluation functions improve the efficiency of search by focusing

attention on the most promising states. Confusingly, both strategic rules and evaluation functions are called heuristics in the literature (Pearl, 1984). Generative selectivity resides in the procedural knowledge whereas evaluative selectivity resides in the principled knowledge. The production rules generate actions, and the state constraints evaluate the states produced by those actions. The performance of the system is a function of both, and one type of selectivity can be traded for the other.

2.6 Summary

This chapter provides rationales for heuristic search as a constraint of PSCD. It is shown that heuristic search is the fundamental process of an intelligent system. The rationales for heuristic search are based on three hypotheses. First, the physical symbol system hypothesis states that all intelligent system belong to the class of physical symbol systems. Second, it is hypothesized that all physical symbol systems represent problems or tasks as problem spaces. Third, all physical symbol systems have limitations in processing capabilities, i.e., have bounded rationality. Integrating these hypotheses yield the heuristic search hypothesis which states that all physical symbol systems exercise intelligence by heuristic search through problem spaces.

Designing a heuristic search system intended to show intelligent behaviors such as that demonstrated by a human is not simple. A notorious problem lies in how to represent knowledge (Barr and Feigenbaum, 1981). The research area of knowledge representation has a long, complex, and as yet non-convergent history (Brachman and Levesque, 1985). This is basically nonidentifiability problem, i.e, difficulties in finding an appropriate

representation and process. Another problem in developing a heuristic search system is the *irrelevant-specification problem* (Pylyshyn, 1984; VanLehn, 1984; Kieras, 1985; Neches, 1982). Because a cognitive model attempts to emulate humans' cognitive processes, the computer program of the cognitive model will contain codes motivated by psychological assumptions about the humans' cognitive processes. The computer program may also contain codes which are not motivated by psychological assumptions but are written for convenience. When the computer program includes the convenience code, the problem is that, because of the interaction between the motivated code and the convenience code, one has no guarantee that the behavior of the program depends on the motivated code. A solution to this irrelevant-specification problem as well as the nonidentifiability problem is to distinguish an architecture from a program (Newell, 1973; Newell, 1990). An architecture is the fundamental design specification of an information processing system, and is based on psychological hypotheses. The architecture provides psychologically motivated programming language. Therefore, a program written in the architecture may not contain any convenience code.

Chapter 3 THE PRODUCTION SYSTEM CONSTRAINT

Production system architecture is the second constraint for PSCD. A production system can be viewed as a basic architecture which provides a basic structure for more integrated architecture, for instance Newell's SOAR (1990) and Anderson's ACT* (1983). There are several basic cognitive architectures in cognitive science. Among them are schema architectures (e.g., Schank and Abelson, 1977; Riesbeck and Schank, 1989), neural architectures (e.g., Rumelhart and McClelland, 1986), and production system architectures (e.g., Newell and Simon, 1972; Newell, 1973b; Anderson, 1983; Newell, 1990).

This chapter discusses a production system architecture (Section 3.1.1). Production system architectures are described along with schemata architectures, which has been considered as the major alternative (Section 3.1.2), and advantages of production system architectures are discussed (Section 3.1.3). Relationships between production system architectures and heuristic search are discussed in Section 3.2.

3.1 Production Systems as Cognitive Architectures

Production systems have been extensively used in cognitive theorizing (Langley, 1983b; Anderson, 1983; Newell, 1990). Production systems have been a primary cognitive architecture, along with schema theories. In the following subsections, a production system architecture and a schema architecture are compared and discussed in detail.

3.1.1 Production Systems

The basic claim of production systems is that underlying human cognition is a set of condition-action pairs called productions. The condition specifies some data patterns, and if elements matching these patterns are in working

memory, then the production can apply. The action specifies what to do in that state. The basic action is to add new data elements to working memory.

Production systems can be traced back to the proposals of Post (1943), but Post production systems have little in common with current production systems except the notion of condition-action pairs, called productions. Modern production systems began with the work of Newell (1973), Newell and Simon (1972), and Waterman (1970). Newell and Simon (1972) first proposed production systems as one way to formulate information processing theories of human problem solving behavior. They initially used the production system, PSG, to provide theoretical accounts of human performance on a variety of puzzles. Klahr and Wallace (1972) used a production system to model children's responses to class-inclusion questions. It soon became apparent that production systems were well suited for dealing with issues of development (Klahr and Wallace, 1976) and learning (Waterman, 1970). Waterman (1970) implemented an *adaptive production system* to play the game of draw poker. The program was adaptive in that it automatically changed the productions in its rule base. This automatic revision or addition of new rules is viewed as learning. Several, different adaptive production systems have been developed to model human learning (Anderson et al., 1981; Anzai and Simon, 1979; Langley, 1983a; Klahr et al., 1987).

The concept of a production system is vague and hard to define. Computer scientists would regard production systems as a variant of deductive retrieval systems. In computer science there is a more general category, called pattern-directed systems (Waterman and Hayes-Roth, 1978), which include schema systems as well as production systems. To psychologists, production systems could be a formalism to state theories of HIP.

In its most basic form a production system consists of two interacting data structures (working memory and production memory), connected through a simple processing cycle (recognize-act-cycle) (Neches et al., 1987). The following description of a basic production system is taken from Neches et al. (1987). The two interacting data structures are:

1. A *working memory* having a collection of symbolic data items called working memory elements.

2. A *production memory* consisting of condition-action rules called production, whose conditions describe configurations of element that might appear in working memory and whose actions specify modifications to the contents of working memory.

Production memory and working memory are related through the recognize-act-cycle. This consists of three distinct stages:

1. The *match* process, which finds productions whose conditions match against the current state of working memory; the same rule may match against the current state of working memory in different ways, and each such mapping is called an instantiation.

2. The *conflict resolution* process, which selects one or more of the instantiated productions for applications.

3. The *act* process, which applies the instantiated actions of the selected rules, thus modifying the contents of working memory.

The basic recognize-act process operates in cycles, with one or more rules being selected and applied, the new contents of memory leading another

set of rules to be applied. This cycling continues until no rules are matched or until an explicit halt command is encountered.

Suppose there is a production system with one production rule and working memory elements as shown in Figure 3.1. The single production and working memory elements are represented in the OPS5 programming language (Brownston et al, 1985). The rule, FindAncestors, finds and prints the ancestors, matching and firing so long as there are still Request elements and matching people. The rule should terminate when there are no longer any matching people to print as ancestors. An English description of the rule is as follows:

IF there is a request to find the ancestor of <myparent> (not nil)
  and there is a person whose name is <myparent>
THEN
  remove the Request
  and print a message that the <mother-name> and <father-name> of <my-parents> are ancestors through <myparent>
  and create two new Requests to find the ancestors of the <mother-name> and the <father-name>

Figure 3.1. An example of a production system

< Production memory >

```
(p FindAncestors
    {(Request ^type ancestor ^target {<myparents> <> nil}) <request1>}
    (Person ^name <myparents> ^mother <mother-name> ^father <father-
        name>)
-->
    (remove <request1>)
    (write (crlf) <mother-name> and <father-name> are ancestors via
        <myparents>)
    (make Request ^type ancestor ^target <mother-name>)
    (make Request ^type ancestor ^target <father-name>)
```

< Working Memory >

```
1: (Person ^name Penelope ^mother Jessica ^father Jeremy)
2: (Person ^name Jessica ^mother Mary-Elizabeth ^father Homer)
3: (Person ^name Jeremy ^mother Jenny ^father Steven)
4: (Person ^name Steven ^mother Loree)
5: (Person ^name Loree ^father Jason)
6: (Person ^name Homer ^mother Stephanie)
7: (Request ^type ancestor ^target Penelope)
```

Let us consider what will happen when we start the production system running with the two production rules in Figure 3.1. Matching the production FindAncestors to working memory elements results in an instantiation with variable bindings ((<name> Penelope)(<mother-name> Jessica)(<father-name> Jeremy)). Firing the instantiation removes the element (Request ^type ancestor ^target Penelope) and adds two elements to working memory: 8: (Request ^type ancestor ^target Jessica) and 9: (Request ^type ancestor ^target Jeremy).

At the next cycle, there are two instantiations of the rule FindAncestors: one matching Jeremy and one matching Jessica. One of the most common conflict resolution strategies is recency, in which an instantiation with the most recent element in working memory is selected for execution. With the recency strategy, the instantiation matching the Request for the ancestors of Jeremy will be selected because it matches the most recent working memory element.

The basic production system consists of a working memory, a production memory, and conflict resolution. Within this basic framework of a production system, there can be many variations. There is a class of production systems called *adaptive production systems* (Waterman, 1975; Neches et al., 1987). Adaptive production systems contain a learning mechanism as well as the basic components of production systems in order to model human learning and development. Another class of production systems is *controlled production systems* which allow control knowledge to be directly represented using a control language (Georgeff, 1982). Still another class of production systems utilizes meta-level knowledge (Davis and Buchanan, 1984; Clancey, 1983). These various types of production systems will be discussed in relation with search in Section 3.2.

3.1.2 A Schema Architecture

Production systems are not the only kind of cognitive architectures since generality and computational universality can be achieved in very different architectures. The most commonly offered alternatives to production systems are the *schema* architecture (e.g., Minsky, 1975; Schank and Abelson, 1977; Rumelhart et al., 1986). The term schema is used more by psychologists and cognitive scientists than by AI researchers. The schema architecture view intelligence as being based on the associationistic properties of knowledge, i.e., the units of knowledge tend to be not elementary facts but cohesive clusters of related facts. In AI, terms such as frame and script tend to be used more frequently. Partridge (1991) commented that a schema may be more of an abstract knowledge representation unit, while frames and scripts tend to be more concrete, even if they are not always part of an implemented and working system. The term schema will be used in this study to refer to all of forms of structured representation, including frames, scripts, and schemata.

The two major components of the schema framework are a working memory and a schema memory. The major control cycle of the system involves matching schemata to the contents of working memory. A schema can be a fancy record structure, such as in frame. A frame is a collection of data items that possess some similarity. For example, a chair record might be a collection of data items such as color, number of legs, height, etc. In addition the chair frame contain links to more general concepts such as furniture. The chair frame will also have links to more specialized frames such as an arm-chair frame. This type of link is called *isa*, which says that one class is a more general version of another. If we want to denote a particular chair, say black chair at my office, we can construct a frame and name it, say chair-1. The chair-1 frame

also can be linked to the chair frame. This type of link is denoted as *inst*, which says that a particular individual is a member (or instance) of some class. If, when constructing the chair-1 frame, there is no explicit statement of how many legs this particular chair has, then a default value of four in the chair frame can be used to fill in the number-of-legs slot of the chair-1 frame. Thus, in general, if a schema is partially matched by the information in working memory it will create further information to complete this match. It is frequently proposed that an instantiated schema like chair-1 can be deposited in schema memory and then serve as a new more specific schema. An object is recognized by having its schema match the representation in working memory. Thus, recognition is a basic force.

This schema structure would lead to a computationally universal system, so it is certainly capable of producing humanlike cognition. Moreover, a number of empirical phenomena are suggestive of schema operation (Minsky, 1975). Despite all of these contentions, schema theories have many unsatisfactory properties. Anderson (1983) pointed out three problems associated with schema theories. First, the schema structure blurs the distinction between procedural and declarative knowledge. It has been argued that the procedural-declarative distinction is fundamental in many psychology studies (Anderson, 1983; Ohlsson and Rees, 1991; Hiebert and Lefevre, 1986). in addition Anderson (1983) argued that the distinction is useful in capturing the conditional directionality of human thinking. For example, from the fact that the light is green one wants to infer that one can walk. Using the schema architecture to represent the above fact risks the incorrect inference that the light is green from the fact that one is walking. This incorrect inference is possible in schema theories because it is possible to instantiate any part and execute any

other part. In other words, schemata are symmetric. In a production system, on the other hand, this asymmetric conditionality can be easily achieved by the condition-action productions, that is, one can only go from the instantiation of the condition to the execution of the action, not from the action to the condition. In general, declarative knowledge is more flexible but inefficient, whereas procedural knowledge is less flexible but efficient because it captures efficiency in its structure and direction of information flow. Schemata are more like declarative and are flexible. However, as mentioned above, although the symmetry of schemata may be a source of flexibility, it can lead to problems such as incorrect inference.

A second problem is that the units of knowledge in the schema architecture tend to be large, thus not allowing the ability to combine information in new ways from smaller units. This ability is argued as a hallmark of human intelligence (Anderson, 1983). Lastly, the size of schemata also makes it difficult to construct effective theories about their acquisition. Technically, it is difficult to construct learning mechanisms that can deal with the full range of schema complexity. Empirically, it is transparent that learning is gradual and does not proceed in schema-sized jumps. This has led schema theorists (Rumelhart and Norman, 1981) to propose that we never learn new schemata but only modifications of existing ones.

It is possible to model systematic errors with the schema architecture. In the schema architecture, errors can arise (a) from fitting the data to the wrong schemata, (b) from matching the correct schema too enthusiastically so that unmatched portions are filled with best guesses rather than available information, and (c) from relying too heavily on active or salient schemata (Reason, 1990). Most of these errors can happen because of a property

associated with schemata theories: a schema only contains information of how a particular recollection of information should appear, but has no information of what it should not look like. It is intuitively clear that people frequently recognize events by identifying that they belong to a generic concept that represents them. It may also be the case that people's belief about what a concept does not look like do help prevent them committing errors. What is missing in schemata theories is such a mechanism.

3.1.3 Why Production Systems?

The production system architecture has been used extensively in cognitive science studies (Newell and Simon, 1972; Anderson, 1983; Newell, 1990; Neches et al., 1987). There are many good reasons for taking the production system architecture as a good framework for modeling human thinking. Neches et al. (1987) lists six advantages of production system models:

> *Homogeneity.* Production systems represent knowledge in a very homogeneous format, with each rule having the same basic structure and carrying approximately the same amount of information. This makes them much easier to handle than traditional diagram.
>
> *Independence.* Production rules are relatively independent of each other, making it easy to insert new rules or remove old ones. This makes them very useful for modeling successive stages in a developmental sequence and also makes them attractive for modeling the incremental nature of much of human learning.
>
> *Parallel/Serial nature.* Production systems combine the notion of a parallel recognition process with a serial application process; both features seem to be characteristic of human cognition.

Stimulus-response flavor. Production systems inherit many of the benefits of stimulus-response theory but few of the limitations, since the notions of stimuli and responses have been extended to include internal symbol structures.

Goal-driven behavior. Production systems can also be used to model the goal-driven character of much of human behavior. However, such behavior need not be rigidly enforced; new information from the environment can interrupt processing of the current goal.

Modeling memory. The production-system framework offers a viable model of long-term memory and its relation to short-term memory, since the matching and conflict resolution processes embody principles of retrieval and focus of attention.

Anderson (1987) argued for the production system architecture in modeling skill acquisition. He observed that production rules are relatively well structured, simple and homogeneous, and independent of one another. In being well structured, they contrast with theoretical formalisms such as neural models (McClelland and Rumelhart, 1986). This helps guarantee that the behavior produced by learning production rules will be coherent. In being simple and homogeneous, they contrast with many of the proposals for schema systems (Schank and Abelson, 1977). Their simplicity and homogeneity make it possible to define relatively simple learning mechanisms capable of constructing new rules. In being independent, they contrast with a typical programming language in which operations are sequentially ordered and depend on one another. In contrast, it is possible to add or delete production rules individually without greatly perturbing the system. This independence of production rules makes it possible to define an incremental learning system that

grows one production rule at a time and does not involve wholesale changes to the cognitive procedures.

Viewed from the perspective of cognitive diagnosis, production systems have a better capability of modeling errors at a fine-grained level than do schemata theories. In production systems, errors can occur when a production fails to fire when it should (errors of omission) or a production fires when it should not (errors of commission) (Newell and Simon, 1972; Bundy and Silver, 1982). An error of commission occur when a production is overly specific and does not match even when it should. Such a production system will be conservative. It would make no bad moves and would miss some good moves. An error of omission would occur when a production is overly general and matches situations when it should not. The production system will be a rash one, omitting few desirable moves but considering many undesirable ones as well. Production systems thus are able to model a broad spectrum of errors, from conservative to rash.

3.2 Search and Production Systems

From the perspective of problem space and heuristic search, problem solving in production systems can be viewed as attempts to find a path linking the initial state to the goal state (Newell and Simon, 1972; Anderson, 1983). Actually production systems are computationally universal and hence can be used to model any computable procedure, including search. Since problem solving is viewed as searching for a path through a problem space from an initial state to a goal state, the process of solving the problem can be modeled as a production system (Rich, 1983) Three components of a search system (i.e., database, operators, and control) are well mapped into three components of a

production system (working-memory, production memory, and conflict resolution). A database is analogous to a working-memory, operators to productions and control to conflict resolution.

As in a search system, a production system needs knowledge to avoid the problem of combinatorial explosion. In a production system, knowledge can be put in various ways. Knowledge can be domain-dependent or domain-independent. Domain-dependent knowledge can be represented in the form of conditions of productions (Langley, 1985). This representation will increase generative selectivity. Another way of representing domain-dependent knowledge uses an augmented mechanism. For example domain-dependent knowledge can be represented in state constraints, an augmented mechanism to a basic production system, that will increase evaluative selectivity (Ohlsson and Rees, 1991). Domain-dependent knowledge also can be represented as meta-rules, i.e., rules about rules, that will increase generative selectivity. Domain-independent knowledge ranges from as simple as weak-methods (Newell, 1969), to conflict-resolution principles (McDermott and Forgy, 1978), and to meta-strategies (Clancey, 1983). These various methods are discussed below.

Depth-first search starts at the root of the problem-space tree and work down the leftmost branch to the end-node before embarking on any other branch. Depth-first search does not guarantee to identify the shortest path to the solution. In breadth-first search, all of the nodes in the problem-space tree at depth 1 are developed first, then all of the nodes at depth 2 are developed and so on. Breadth-first search overcomes the problem associated with depth-first search and finds the shortest path solution. However, the breadth-first search can involve huge overhead: at any stage, all nodes to the left and all

nodes above the node being developed must be memorized. A more efficient method is heuristic search. Contrary to depth-first and breadth-first search, heuristic search uses heuristics to improve the quality of the paths that are explored. Some heuristics are general in that they are useful in a wide variety of problem domains. Control strategies that use general-purpose heuristics are often called *weak methods*. Weak methods include generate and test, hill climbing, best-first search, and etc. In production systems, analogous to conflict strategies is conflict resolution. McDermott and Forgy (1978) proposed a set of domain-independent conflict resolution principles.

General-purpose heuristics usually are coupled with domain-specific, special-purpose heuristics to work well in a specific domain. Domain specific heuristics can be incorporated into a rule-based search procedure in two major ways: in the rules themselves and as a heuristic function (Rich, 1983, p. 71). A heuristic function is a function that measures desirability, usually represented as numbers, of individual problem states in leading to the goal state. The purpose of a heuristic function, therefore, is to guide the search process in the most desirable direction, by suggesting which path to follow first when more than one is available.

Domain-specific information can be incorporated into rules. One way is to devise rules about rules (Davis et al., 1977; Davis and Buchanan, 1984). Davis et al. (1977) represented certain types of heuristic search strategy knowledge as *meta-rules*. Figure 3.2 shows an example of such meta-rules. Such meta-rules, which are invoked as part of the conflict resolution strategy, can capture and implement strategic knowledge about a domain. Clancey (1983), however, pointed out that meta-rules such as the one above have the disadvantage that they mix the domain-dependent and domain-independent

strategic principles that underlie them. The implicit strategic domain-dependent principle which underlies the rule above is that the most frequent causes of a disorder should be considered first. The implicit domain-independent principle which underlies the rule above is that the most useful rule should be applied first. Clancey (1983, 1988) argued that different types of knowledge must be represented separately and explicitly in order to provide sensible explanations to users.

Figure 3.2. An example of meta-rules

If the infection is pelvic abscess and
  there are rules which mention enterobacteriacea in their premises and
  there are rules which mention gram-pos-rods in their premises
Then there is suggestive evidence (0.4) that the former rules should be applied
  before the latter.

Georgeff (1982) has proposed a *controlled production system* architecture that allows control knowledge to be directly represented using a control language. The architecture is based on the explicit specification of constraints on rule invocation. In Georgeff's scheme, control knowledge is represented by a language, with which we can specify the set of all allowable sequences of rules. Georgeff calls this language a "control language." For example, suppose that the control language is defined by the following regular expression: R1 (R2R3R4)* R5R6R7 where the asterisk indicates repetition of the bracketed group. The only allowed rule sequences would be: R1, followed by the sequence R2R3R4 repeated an arbitrary number of times, followed by the sequence R5R6R7. The restriction imposed by the control language can be

invoked either when the set of applicable rules is being identified or as part of the conflict resolution strategy.

Another way of incorporating domain-specific information to rules is to find appropriate conditions for rules and have them make sensible moves. It has been argued that finding those conditions is relatively hard and that we can use some learning mechanisms to identify those conditions (Langley, 1985). The class of production systems that use learning mechanisms to gain powerful knowledge about the problem which would reduce search efforts are called *adaptive production systems*. Since these systems usually aim at learning search strategies, they are also called *strategy learning systems* (Langley, 1983a, 1985). Strategy learning systems will be discussed in detail in Chapter 5.

3.3 Summary

This chapter discusses a production system architecture and argued for a production system architecture as a sound basis for PSCD. Three problems of schema architectures are discussed along with capabilities of a production system architecture. A production system framework is particularly advocated due to its superior abilities in modeling errors and learning. These abilities are necessary for cognitive diagnosis.

Chapter 4 THE CONSTRUCTIVIST CONSTRAINT

The final constraint that are built into PSCD is the notion of constructivist. The constructivist assumption is a major hypothesis in many cognitive, developmental, and educational studies. The constructivist assumption differentiates two types of knowledge: procedural knowledge and conceptual knowledge (Hiebert and Lefevre, 1986). Procedural knowledge consists of a collection of procedures, and conceptual knowledge consists of a set of principles. Procedural knowledge represents the type of knowledge executable easily, and its execution generates observable performance (Figure 4.1). Conceptual knowledge, on the other hand, represents the type of knowledge that can not be easily executable, but typically be incorporated or constructed into procedures.

Figure 4.1. Principles, procedures, and performances

In the constructivist view, observable performance result from executing a procedure, which in turn is based on principles of conceptual knowledge. The constructivist assumption states that new knowledge is not directly absorbed but is learned through the construction or integration of new knowledge with previous knowledge (Resnick, 1983). It is also found that people tend to invent a procedure from what they know and use that procedure to act on the problem at hand (Resnick and Gelman, 1984). Errors are viewed resulting from the use of such procedures constructed in hasty.

The notion of constructivism is of importance in this study because it identifies the source of errors as the construction process of procedures. This

chapter describes the constructivist constraint of PSCD in detail. The distinction between procedural and conceptual knowledge and the notion of constructivism are discussed in Section 4.1. Section 4.2 summarize the discussion.

4.1 Principles and Procedures

As stated earlier, the distinction between principles and procedures is important in the constructivist assumption. Similar distinctions to the distinction of principles and procedures can be found in many other disciplines. For example, in linguistics, Chomsky (1967) distinguished between competence and performance, and in cognitive development, Piaget (1978) distinguished between conceptual understanding and successful action. In cognitive psychology, Tulving (1983) distinguished between semantic memory and episodic memory; and Anderson (1976) distinguished between declarative and procedural knowledge. In the context of mathematics education, VanLehn (1986) distinguished between schematic and teleologic knowledge; Hiebert and Lefevre (1986) distinguished between conceptual knowledge and procedural knowledge; Resnick (1982) talked about semantics and syntax; Gelman and Gallistel (1978) distinguished between principles and skills; and Baroody and Ginsburg (1986) described differences between meaningful and mechanical knowledge.

Hiebert and Lefevre (1986) observed that underlying these various types of knowledge was the distinction between skill and understanding, and argued that the distinction of conceptual and procedural knowledge drew upon all of them. In other words, it is this distinction of skilled procedures and understanding of principles that underlies those various types of knowledge.

According to Hiebert and Lefevre, conceptual knowledge is knowledge that is rich in relationships and is thought of as a network in which facts and propositions are linked with each other. Hence, a unit of conceptual knowledge cannot be an isolated piece of information. It is a part of conceptual knowledge only if the holder recognizes its relationship to other pieces of information.

Procedural knowledge, on the other hand, consists of knowledge of forms and knowledge of procedures. Knowledge of forms is a familiarity with the individual symbols of the system and with the syntactic conventions for acceptable configurations of symbols. For example, those who possess knowledge of forms would recognize that the expression 3+5 = (?) is syntactically acceptable and that 3+ = (?) is not acceptable. Knowledge of procedures or rules specifies a predetermined sequence of execution on symbolic (e.g., +,-) or nonsymbolic (e.g., concrete objects or mental images) objects. Hiebert and Lefevre (1986) suggested that procedure knowledge can be characterized as production systems (Newell and Simon, 1972) in that for the completion of a task a set of productions or rules are applied to an initial state to produce a goal state in a sequential manner. The insight of Hiebert and Lefevre (1986) appears to be that the primary relationship in procedural knowledge is "after," which is used to sequence subprocedures and superprocedures linearly while conceptual knowledge possesses many kinds of relationships.

Given these two types of knowledge, an important question becomes relationships between conceptual and procedural knowledge. Determining relationships between conceptual and procedural knowledge is not simple. A persistent problem is that conceptual knowledge is difficult to measure directly

and consequentially is often inferred through the observation of particular procedures for which it presumably is a prerequisite.

Carpenter (1986) proposed three possible relationships. The first hypothesis is that advances in procedural knowledge are driven by broad advances in conceptual knowledge. The second is that advances in conceptual knowledge are neither necessary nor sufficient to account for all advances in procedural knowledge. The third hypothesis agrees with the first in that advances in procedural skills are linked to conceptual knowledge, but proposes that the connections are more limited than suggested by the first.

Supporters of the first approach are Riley et al. (1983) and Briars and Larkin (1984). They attributed development to broad advances in conceptual knowledge and the linking of procedures to this conceptual knowledge. Baroody and Ginsburg (1986), however, argued that development of procedures used to solve addition and subtraction problems was not always governed by the development of conceptual knowledge, but the application of certain procedures may lead to conceptual knowledge. Brown and Burton (1978), Brown and VanLehn (1982), and VanLehn (1986) also argued for nonnecessity of conceptual knowledge for the development of procedural knowledge. They argued that children could learn procedures by rote without relating them to any conceptual knowledge, and some invention in procedures appeared to occur strictly within the context of procedural knowledge. They therefore interpreted children's arithmetic errors as resulting from small perturbations of syntax of arithmetic without reference to semantics (i.e., conceptual understanding of principles) of arithmetic. This perspective, however, has undergone harsh criticism. Prominent opponents to this view of no relationship of conceptual knowledge to the development of procedural

knowledge are Resnick (1982, 1983, 1984, 1989) and Gelman and Meck (1983, 1986).

Gelman and Meck (1983, 1986) argued that some principled understanding precedes skilled counting both because the data support this view and because if there were no early competence, children might never learn to count or do arithmetic. Reviewing relevant research on children's knowledge and learning of mathematics, Resnick (1983) and Resnick and Gelman (1984) argued that underlying diverse research done by cognitive, developmental, and educational psychologists was a so-called *constructivist* assumption about how mathematics was learned.

The constructivist assumption states that knowledge is not directly absorbed but is constructed by each individual. In this view, knowledge is no longer viewed as a reflection of what has been given from the outside; it is a personal construction in which the individual imposes meaning by relating bits of knowledge and experience to some organizing schemata. Here schemata refers to any interpretive structures brought to the problem (Resnick and Gelman, 1984). It appears that the term schemata used by Resnick (1983) refers to a higher concept than the terms like schemata theories which are meant to include frames, scripts and any other structured object representations.

It is viewed that people always seem to try to make sense out of the world, and to create rules for acting in it, even given limited information. They do not wait until all the information is in before they start to construct a "theory" to account for what they have before them. In other words, they invent a procedure from what they know and use that procedure to act on the problem at hand. It has been shown that implicit in invented procedures is conceptual

understanding of mathematical principles (Gelman and Meck ,1983, 1986; Resnick and Neches, 1984). Resnick (1987) and Resnick and Omanson (1987) provided data which suggested that arithmetic errors by children resulted from their attempts to invent a procedure which respected all the information they did have while ignoring a mathematically important constraint that was apparently not learned.

Gelman and Meck (1986), in addition, argued that knowledge of the correct principles does not guarantee correct performance. It was hypothesized that the performance structures are consequences of competence structures, derived by a planning system (Greeno et al.,1984). In their formulation, competence has three main knowledge sources: conceptual, procedural, and utilizational competence. Conceptual competence is the implicit understanding of general principles of the domain. Procedural competence is understanding of general principles of action and takes the form of planning heuristics. Utilizational competence is understanding of relations between features of a task setting and requirements of performance. In this formulation, performance depends on procedural and utilizational competence as well as conceptual competence. Thus, connections between conceptual and procedural knowledge are established gradually and locally to individual problems on a piecemeal basis. This conclusion is argued to be consistent with more general findings from research on cognitive development (Gelman and Meck, 1986; Carpenter, 1986) and also appears to be consistent with recent studies in situated cognition (Greeno, 1989). Recently, Smith et al. (1989) built a computational mechanism, called COUNTPLAN, based on the work of Greeno et al. (1984).

More recently, Ohlsson and Rees (1991) proposed a cognitive mechanism that mediates the facilitation of conceptual knowledge to procedure acquisition. This mechanism is based on the assumption that conceptual knowledge facilitates procedure learning. Ohlsson (1986) referred to such mechanisms that derive procedures from conceptual knowledge as "rational" learning to distinguish them from "empirical" learning which constructs procedures on the basis of experience. Ohlsson's (1991) rational learning mechanism consists of two main components: a representation for conceptual understanding and computational machinery that maps that understanding onto a procedure. Ohlsson and Rees (1991) proposed a novel knowledge representation, called *state constraints* to represent principled understanding. Since the computational framework of the state constraint formalism is heuristic search, the state constraint formalism is of the most relevance to this study. In the state constraint formalism, principles are represented as state constraints, and those state constraints act as constraints on possible states of affairs during the search. The choice of the state constraint theory for PSCD in this study is due to the heuristic search and production system constraint s. Clarity and parsimony of the state constraint theory also merits attention in this choice process.

4.2 Summary

In this chapter, it is argued that the distinction between principled understanding and procedural knowledge is fundamental in many distinctions of knowledge found in many other disciplines. Of more importance is the relationship between two types of knowledge. It is argued that new knowledge of procedures is not directly absorbed but is learned through the construction

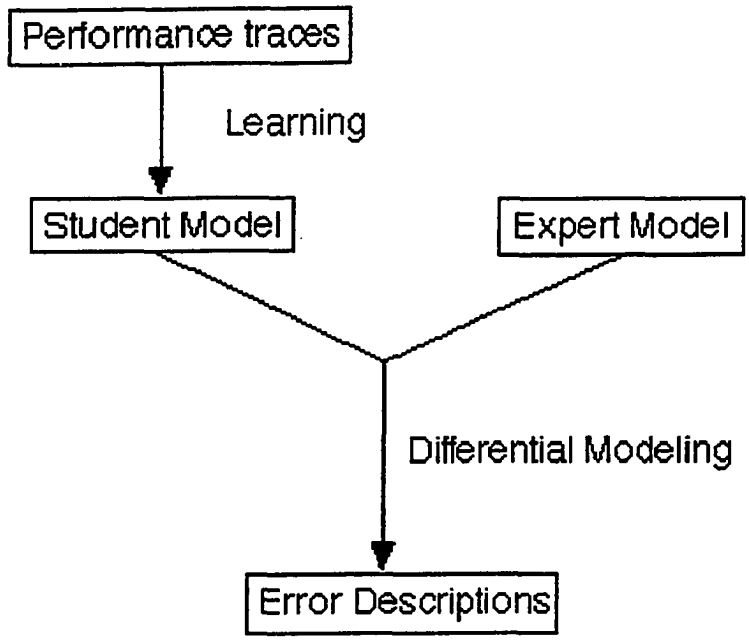
process. Understanding of domain principles aids the construction process. Thus, errors are viewed as arising from using incorrect procedures that are constructed with incomplete understanding. Incorrect procedures are not just variants of correct procedures but have roots in incomplete understanding of domain principles. Errors, procedures, and the role of domain principles in procedure construction are discussed in detail in the next chapter.

Chapter 5 METHODOLOGIES OF COGNITIVE DIAGNOSIS

In Chapters 2,3, and 4, three constraints that are incorporated into PSCD are described. This chapter reviews different methodologies of cognitive diagnosis and describes relationship with three constraints embedded in PSCD. Cognitive diagnosis refers to the process of inferring a person's cognitive state from his performance (Ohlsson, 1986a; Wenger, 1987). The task of cognitive diagnosis usually involves the construction of a student model (Figure 5.1). The purpose of the construction is to discover which knowledge, correct or incorrect, has been used to produce behavior, and which relevant knowledge has been overlooked. The process of discovering differences of a student model from an expert model is called *differential modeling* (Wilkins et al., 1988). Differential modeling typically produces error descriptions. Error descriptions provide the basis for the potential remedial instruction. On the basis of this belief, several computerized diagnostic methods for identifying the errors of individual learners have appeared in various domains: arithmetic (Burton, 1982; Ohlsson and Langley, 1988), algebra (Sleeman, 1982), elementary programming (Johnson, 1986), and LISP programming (Anderson and Reiser, 1985; Anderson et al., 1984).

Recent studies in ITS progress toward building a program that can understand what the student is doing (Burns et al., 1991; Clancey, 1987b; Mandl and Lesgold, 1988; Polson and Richardson, 1988; Sleeman and Brown, 1982). This progress was the result of realization that there was no need to consider strategies for teaching if we did not understand what the student was doing (Clancey, 1987). It became the central idea to model the student. This realization actually called for a detailed model of the student, that is a fine-

Figure 5.1. Cognitive diagnosis



grained description of the student than such a coarse-grained model as a performance measure. An assumption is that tutoring based on on fine-grained student models will be more effective than tutoring based on coarse-grained models. Although this assumption has not been attempted to check (VanLehn, 1988), it is widely accepted by educational researchers (Self, 1974; 1978; Ohlsson, 1986a; Resnick, 1984) and in recent ITS studies (Clancey, 1987; Anderson, 1988; Johnson, 1986). In Chapter 1, cognitive simulation is advocated as a better approach in modeling a decision maker or student because of its ability to produce detailed models of a decision maker. ITS studies also defend the need of a fine-grained student model.

This chapter discusses different methods of cognitive diagnosis. A cognitive diagnosis method can be understood as consisting of two processes: first inferring a student model from performances traces and second producing error descriptions by comparing a student model and an expert model. ITS literature suggest two different methodologies for the inference of a student model: machine learning and planning. In this study, the focus is placed on studies incorporating models of learning into cognitive diagnosis. Underlying the focus is the belief that a model of cognitive diagnosis would be more useful if it were built on learning theories. This belief is in line with studies in development and education, for instance by Glaser and Bassok (1989). It is argued that a theory of instruction must specify itself in terms of three dimensions: (a) the knowledge state of competent performances ; (b) the initial knowledge state of the learner ; and (c) the process of learning, the transition from initial state to desired state that can be accomplished in instructional settings (Glaser and Bassok, 1989). The insight of Glaser and Bassok is that it is necessary to consider the above three dimensions in order to determine the

best remedial instruction out of possible instructions. What it implies in the context of ITS is that the tutoring model must have information on these three dimensions. In other words, the model of cognitive diagnosis must generate such necessary information with a student model and a expert model. Thus, various cognitive diagnosis methodologies are discussed in terms of these three dimensions.

Section 5.1 reviews several studies concerning descriptions of in knowledge states between an expert and a student. Three approaches are described, and among them the bug part approach which reconstruct buggy rules is favored as an appropriate approach for this study (5.1.1). Planning and learning are two methodologies for reconstructing buggy rules, and the learning approach is advocated (5.1.2). Section 5.2 reviews relevant learning theories. Learning can be viewed as strategy learning in production system frameworks (5.2.1). Two types of techniques frequently used in strategy learning studies are generalization and discrimination (5.2.2). It is shown that discrimination learning has potential to overcome some of disadvantages of generalization learning (5.2.3). Empirical learning has been the dominant paradigm in machine learning, but it is argued that empirical learning models are not appropriate for the purpose of cognitive diagnosis (5.3). It is argued that domain principles are important in procedure construction (5.4), and rational learning models are favored for the cognitive diagnosis purpose (5.5). Finally the summary of this chapter is provided in Section 5.6.

5.1 Differences between Expert and Student

Cognitive diagnosis can differ in terms of a knowledge representation scheme employed to express knowledge states of a competent expert and a

novice student. We may think of two different representation schemes, one for a competent expert and the other for a novice student. However, most cognitive diagnosis studies assume one representation primarily because of efficiency and economy (VanLehn, 1988). Typically one representational language is used to represent knowledge states ranging from the initial to the competence. Novices are usually viewed lacking some knowledge and/or keeping incorrect knowledge as compared to experts. Therefore, a student model can be more or less a subset of a expert model and/or a expert model with incorrect knowledge. Literature shows three types of approaches: the overlay approach, the bug library approach, and the bug part approach (Wenger, 1987; VanLehn, 1988).

In the overlay approach, a cognitive diagnosis model is concerned with only missing knowledge (Clancey, 1987). Conceptually, a student model is a proper subset of a expert model. A student model consists of the expert model plus a list of items that are missing. In the bug library approach, both incorrect knowledge and missing knowledge are represented (Anderson et al., 1990; Sleeman, 1984). A student model here consists of an expert model plus a library of predefined misconceptions and missing conceptions. The members of this library are called bugs. In the bug part library, ITSs construct bugs from a library of bug parts. Bugs are constructed during diagnosis rather than being predefined. For instance, each bug constructed by the ACM system (Langley and Ohlsson, 1984) is a production rule which consists of conditions, that are learned during diagnosis, and single action. The action and the predicates used for inducing conditions are drawn from the defined representation language.

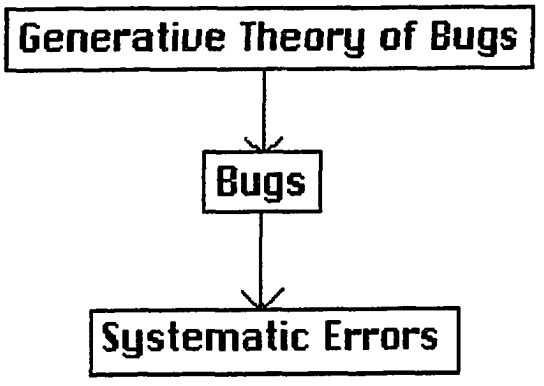
Among three approaches, the overlay approach is the most frequently used in representing student models. The overlay approach, however, is not

able to model incorrect knowledge which was found to account for many of the errors that have been shown in children's subtraction (Matz, 1982; Brown and VanLehn, 1980; Young and O'Shea, 1981; Stevens et al., 1982). The bug library approach could be better in accounting for errors because it can model errors resulting from incorrect knowledge as well as those from missing knowledge. The biggest hurdle in the bug library approach is in assembling the library (VanLehn, 1988).. The library should be nearly complete. If a student has a bug that is not in the library, then the student model will try to fit the behavior with some combination of other bugs. It may totally misdiagnose the student's misconceptions. It is the bug part approach which can account the largest set of errors. This is so because bugs are constructed rather than predefined. Bugs can be constructed through a planning technique (Kowalski and VanLehn, 1988; VanLehn and Garlick, 1987) or through a machine learning technique (Ohlsson and Langley, 1988; Langley et al., 1990).

5.1.1 Generative Theories of Bugs

So far discussed are how much each approach is able to account for errors. What is obviously missing in the discussion is why errors occur in the first place, in other words explaining the cause of errors. It is useful to distinguish a generative theory, bugs, and systematic errors (Figure 5.2).

Figure 5.2. Generative theory of bugs (Brown and VanLehn, 1980, p. 380)



Errors are said to be *systematic* if there exists a procedure that produces his erroneous answers. Systematic errors occur from using erroneous variations of a correct procedure for that skill. These erroneous variations of a procedure are called *bugs* or *mal-rules*. Bugs are viewed as complex, intentional actions reflecting mistaken beliefs about the skill, and are differentiated from slips where the subject did something which they did not intend to do. Bugs can be collected empirically from examining protocols and added to a correct procedure to diagnose students' behavior (Brown and Burton, 1978; Sleeman, 1984). Or bugs can be reconstructed via a computational mechanism (Brown and VanLehn, 1980; Young and O'Shea, 1981). Brown and VanLehn posited that there would be a small set of principles and associated processes that could transform a correct procedural skill into all of its buggy variants. Thus, Brown and VanLehn hypothesized that all observed errors could be explained by buggy variants which in turn could be generated by applying one or more hypothesized processes to a correct procedural skill. Hence those principles and processes are called a *generative*

theory of bugs. In this study, three generative mechanisms are reviewed: repair (Brown and VanLehn, 1980), deletion (Young and O'Shea, 1981), and misgeneralization (Matz, 1982; Sleeman, 1984).

Brown and VanLehn (1980) proposed a generative theory of bugs, called *repair* theory, which generated all the known or expected bugs for a particular skill, in this case subtraction. According to this approach, bugs originate mostly in mislearning and forgetting: that is, in the student's failure to follow an instructional sequence or in the teacher's failure to provide an unambiguous and organized set of instructive examples. The essential idea of the repair theory is that some fault in a student's core procedure for subtraction leads to *impasses*, which are problem states from which the core procedure is unable to proceed. When this happens a local problem solver is brought into play to devise a *repair* to bridge to a new problem state from which the knowledge base can take over. The observed bugs thus arise from the cross product of the set of possible impasses with the set of possible repairs.

Another generative mechanism that explains the source of errors is *deletion* (Young and O'Shea, 1981). Similar to Brown and VanLehn (1980), Young and O'Shea (1981) argued that many errors in children's subtraction resulted from the use of incorrect strategies rather than from the incorrect recall of number facts. They used a production system to represent the procedure for correct subtraction and argued that many different kinds of errors could be accounted for mainly by the omission of various rules from the set of correct productions.

Matz (1982) and Sleeman (1984) proposed *misgeneralization* as the main source of errors. Matz (1982) argued that errors could be viewed as the result of a systematic adaptation of previously acquired knowledge using a

small number of extrapolation technique. Sleeman (1984) similarly argued that many of the bugs encountered in the subtraction domain could be accounted for by the process of misgeneralization. As contrary to repair theory which hypothesizes that bugs occur because the student has not encountered the appropriate teaching necessary to perform the task, misgeneralization assumes that the student actively tries to infer procedure with his previous knowledge of the domain and errors occur because the procedure cannot be correct due to some missing knowledge. In this approach, students are viewed active theory builders. This view complies with the constructive view developed in cognitive development and education studies (see Section 4.3).

All these three approaches attempt to explain why errors occur from different perspectives, but they all posit the existence of bugs that cause errors. Studies under this presumption are therefore often called bug studies. The whole enterprise of bug studies share two assumptions (Payne and Squibb, 1990). One is that errors can be classified as *slips* or *mistakes* (see Section 7.3 for more discussion about slips and mistakes). Slips occur when one fails to execute as intended. Mistakes occur when one formulates the intention incorrectly. Slips are often called execution errors, and mistakes are called planning errors (Kowalski and VanLehn, 1988). Bug studies focus on mistakes. The other is that bugs arise in the context of purely *syntactic manipulations* of symbols without considering any semantic rationalization of these manipulations. A basic insight of bug studies is that children often used systematic routines that produce wrong answers and those systematic routines involve in mainly syntactical manipulations. A challenge to this assumption is the view that bugs can be seen as corresponding to violations of specific principles (Resnick, 1982; Resnick, 1983; Resnick and Omanson, 1987;

Langley et al., 1990; Ohlsson and Rees, 1991). In the latter perspective, bugs are not caused by faulty syntactic manipulations but by faulty understanding of domain principles. Thus, in this perspective, a generative theory of bugs is not like repair, deletion, or misgeneralization but must contain a process of semantic rationalization.

There are two computational mechanisms in ITS literature, that attempted to model a process of semantic rationalization. Smith et al. (1989) built a computational model called COUNTPLAN which generated procedures through a planning process. Another computational model, called HS, was proposed by Ohlsson and Rees (1991). HS used a learning process to generate procedures. It appears that planning and learning are two main alternatives in cognitive diagnosis research attempting to build a generative theory of bugs.

5.1.2 Reconstructing Buggy Rules: Learning or Planning

Generative mechanisms have been advocated because they explain in principle ways why errors occur, and also because they are able to model complex errorful behavior (Young and O'Shea, 1981; Sleeman, 1984). All three generative mechanisms explain, from different perspectives, what causes errors. Repair theory assumes a repair mechanism which patches a constrained procedure at an impasse and explains errors with such patched procedure. Similarly the deletion mechanism explains errors with a partial set of rules, and misgeneralization is proposed as the source of errors, although Sleeman (1984) just proposed the idea of misgeneralization for explaining errors and did not work out any detailed mechanism. All these mechanisms assumes a priori some kinds of computational mechanisms to explain empirically observed errors. On the contrary, it is possible to reconstruct buggy

rules from problem solving traces. There are two alternative approaches attempting to reconstruct bugs from problem solving traces.

One approach uses a *learning* mechanism to induce a set of rules that constitutes a student model. For example Sleeman's (1984) misgeneralization assumes a type of generalization learning. Langley and Ohlsson (1984) applied discrimination learning to the construction of ACM. ACM constructs cognitive models from error data in the domain of subtraction problems. Recently Ohlsson and Rees (1991) proposed a computational mechanism of rational learning which can be easily extensible to the task of cognitive diagnosis. Learning mechanisms are discussed in detail in Section 5.2.

The other approach that reconstructs buggy rules from problem solving traces uses a *planning* technique. Intuitively, a plan is a program of action, or sequence of steps, designed to achieve a desired goal. Problem solving of students can be viewed as executing a set of actions that are intended to contribute to the overall goal. Based on this view, it can be further said that a trace of goals and subgoals that a student used in generating a sequence of actions provides an appropriate structure for determining the correctness of a solution. Plan recognition is the inverse of planning. A planner uses facts about its problem area to produce a plan that achieves its goal. A plan recognizer starts with a sequence of actions, reconstructs the problem solver's plan, and thereby infers the underlying beliefs. The object of plan recognition is to infer a plan tree with given leaves. The leaves of the tree are primitive actions, the nonleaf nodes in the tree are subgoals, and the root node of the tree is the overall goal. Links between nodes in the tree represent goal-subgoal relationships. Such a tree is called a plan.

Genesereth (1982) proposed an idea that plans can be viewed as dependency graphs. Dependency graphs explain how a set of actions achieve its goal in terms of the problem solver's beliefs about the problem domain. He further talked about the usefulness of resulting plans in identifying student's misconception and in offering remediation in context. VanLehn (1987) viewed that plan recognition is computationally similar to parsing a string with a context-free grammar. He looked to the parse tree as an explanation for the action sequence. When the parser could not find any tree, his system, called SIERRA, invented one or more new procedures to complete a parse. Thus he called SIERRA's algorithms as learning by completing explanations. VanLehn and Garlick (1987) built CIRRUS which used parsing for plan recognition. Kowalski and VanLehn (1988) used CIRRUS to induce student models from protocol data. Recently VanLehn et al. (1989) showed that minor perturbations of goal selection strategies could account for a great deal of variance in the strategies of subtraction students. A production system model in VanLehn et al. (1989) differentiated two types of productions: preferences, which contains information about goal selection, and primitive actions, which contains operator selection information. They showed that small perturbations on preference productions could successfully reproduce protocols of eight nonstandard students.

Another model which utilizes the planning concept is COUNTPLAN (Smith et al., 1989). COUNTPLAN is based on the work of Greeno et al. (1984) who formally specified numerical competence as consisting of conceptual, procedural and utilizational competence. Conceptual competence is the implicit understanding of general principles of the domain. Procedural competence is understanding of general principles of action and takes the form of planning heuristics. Utilizational competence is understanding of relations

between features of a task setting and requirements of performance. It is assumed that the performance structures are consequences of competence structures, derived by a planning system. The formalism used for planning is planning nets (VanLehn and Brown, 1980). "Planning net" is a formalism for the particular representation of teleologic semantics of a procedure. Teleologic semantics refers to knowledge about the purposes of each of its parts and how they fit together. A planning net provide a formal characterization of how constraints and actions in the domain are integrated with the help of planning heuristics into a valid plan. Greeno et al. (1984) used planning nets to derive plans from schematic representations of domain-specific and general procedural principles, three types of competence. Actually COUNTPLAN is a generative theory of performance, not a theory of cognitive diagnosis. It is not concerned with inferring a student model from problem solving traces, although Smith et al. (1989) suggested that such diagnosis would be possible.

Both planning and learning approaches appear to be promising. This study follows the learning approach because, in addition to cognitive diagnosis, learning is the central problem in many other applications, for example in automating protocol analysis (VanLehn and Garlick, 1987), psychological modeling of skill acquisition (Singley and Anderson, 1989), and knowledge acquisition problem for expert systems (Langley and Simon, 1981). In addition it is argued that learning techniques have more cognitive fidelity than planning techniques (Ohlsson and Rees, 1991). The learning approach also has been advocated by many researchers (Anderson, 1983; Langley et al., 1990; Newell, 1990; Ohlsson and Rees, 1991).

5.2 Learning Theories

Learning theories specify the function and form by which novice knowledge structure is transformed to expert knowledge structure. It is said that one important dimension for understanding cognitive diagnosis methods is learning theories to be assumed (VanLehn, 1988). Furthermore, assuming a certain learning theory often dictates certain types of knowledge representation (Charniak and McDermott, 1985). Recent extraordinary growth in artificial intelligence and its application has produced a remarkable expansion in machine learning as well (Michalski et al., 1983, 1986; Kodratoff and Michalski, 1990; Kodratoff, 1986; Mitchell et al., 1986; Forsyth and Rada, 1986).

Researchers have proposed various categories for classifying learning theories. Learning theories are often categorized by learning strategies, e.g., learning by being told, learning by doing, failure-driven learning, learning by discovery, and many others (Charniak and McDermott, 1985; Michalski et al., 1986; Cohen and Feigenbaum, 1982). Bundy et al. (1985) classified learning theories on the basis of algorithms. This study uses a broader classification scheme similar to Draper's categorization (1987). The scheme differentiates three categories: rote learning, empirical learning, and rational learning.

One can learn without much knowledge. This learning is called "rote learning" (Draper, 1987; Cohen and Feigenbaum, 1982). In rote learning, the learning system does not need to do any processing to interpret the information supplied by the environment. All it must do is memorize the incoming information for later use. For example, memorizing a fact that "Chicago is 100 miles east from Lincoln" is a typical rote learning and results in adding a factual assertion into memory. Although rote learning may pose some psychological

interesting issues, it may not be called learning after all. Rote learning will not be discussed much further.

Empirical learning refers to the induction of rules with the help of domain-independent, syntactic computations. Empirical learning is often called inductive learning because it involves in the induction of rules. The classic name for empirical learning is concept learning (Winston, 1984). Under the name of concept learning, there is a wide class of research efforts (Mitchell, 1977; Anderson, 1976, 1983; Laird et al., 1986; Anzai and Simon, 1979; Langley, 1983a; VanLehn, 1982). Section 4.1.1 reviews some of concept learning models relevant to the study, and discusses concepts underlying the methodologies.

Mechanisms that infer new rules with the help of knowledge of the relevant domain is called knowledge-based or "rational" learning mechanisms, and are distinguished from empirical learning mechanisms (Ohlsson, 1986). Although there can be many variations, two prominent theories In rational learning are explanation-based learning (Minton,1988) and the state constraint theory (Ohlsson and Rees,1991). These learning mechanisms are discussed in Section 4.1.2.

Since a learning system can only exist in the context of some performance framework (Newell and Simon, 1973; Langley, 1983b), it is important what framework a learning system resides in. This study uses production systems framework as the performance framework like many others (Newell and Simon, 1972; Anderson, 1983; Newell, 1990; Langley, 1983b; VanLehn, 1990; Ohlsson and Rees, 1991; Minton, 1988), and therefore learning systems are described within the production systems framework. Learning in the production systems framework can be viewed as the creation of new

condition-action rules or productions. The appropriate actions of these new rules can often be determined rather easily, since analogous actions can be observed in the environment. However, the conditions under which these actions should be applied is seldom so obvious (Langley, 1987). Thus, much of studies in machine learning has focused on discovering heuristically useful conditions on productions for solving problems (Langley et al., 1980; Langley, 1983). The problem of condition finding can be approached from a number of perspectives. The categorization of learning theories employed in this study differentiates between the empirical learning approach and the rational learning approach. Since it is necessary to have a performance system within which a learning system operates, the next section describes learning in the context of heuristic search and production system architectures.

5.2.1 Strategy Learning Systems

From the perspective of search, learning can be viewed as the process by which general but weak methods are transformed into powerful, domain-specific search heuristics. The set of works which focus on this kind of transitions is called "strategy learning" or "learning search control knowledge" (Langley et al., 1980; Neches et al., 1987; Minton, 1988).

It is argued that three components are necessary for a strategy learning system to improve its search strategies with experience (Langley, 1985). First, such a system must be able to search, so that it can generate behaviors upon which to base its learning. Second, the system must be able to distinguish desirable from undesirable behaviors, and to determine the components of the system that were responsible for those behaviors. Finally, the system must able to use this knowledge to modify its search strategies, so that behavior improves

over time. Since search is a performance framework, the other two issues are more relevant to learning studies. The part of program which identifies faulty knowledge or rules is called *critics*, and the part of program which, given the information about desirable and undesirable behaviors, revises faulty knowledge or rules is called *modifiers* (Langley, 1985; Bundy et al., 1985). The former problem of identifying faulty rules is generally known as *credit assignment problem* (Langley, 1985; VanLehn, 1989). The latter problem of revising rules is called revision problem (Ohlsson and Rees, 1991)

Since so much of AI research has revolved around the notion of search, search is the best understood. Many alternative search strategies have been explored, ranging from very general but weak methods, like depth-first and breadth-first search, to much more powerful methods that incorporate knowledge about specific domains. Weak methods are general because they make minimal assumptions about the problem. They provide a way to decide what to do when one does not know what to do. Learning is viewed as the transition from weak, general methods to specific, powerful methods. With experience, one can gain knowledge which may aid him or her to decide precisely what to do without involving in costly search. It is this aspect that strategy learning systems try to model. A strategy learning system starts with some weak search scheme that can be applied to many different domains and improves its search scheme with experience.

Distinguishing desirable from undesirable behaviors and determining the parts of the system responsible for those behaviors has been called the credit assignment problem. One option for the credit assignment problem is to solve a given problem with legal operators given by searching the problem space. Once the solution paths have been discovered, they can be used to assign

credit and blame. Moves leading to states on the solution path are desirable, since they led to a solution while moves going off the path are undesirable, since they led elsewhere. This approach is called *learning from solution paths* (Langley, 1985; Sleeman et al., 1982). This approach is very general and can be used to assign blame and credit to any problem, but when search spaces are so large that other route must be taken, other credit assignment *heuristics* that do not require complete solution paths must be employed to enable learning to occur while the problem is being solved. These heuristics enable the search process to be directed enough to reach the goal state. These heuristics open the way to learning while doing (Anzai and Simon, 1979). Langley (1985) documents this type of heuristics, which include noting loop moves, noting longer paths, and illegal states.

A strategy learning system can alter its search behavior by either discovering numerical *evaluation function* or by modifying heuristics with symbolic conditions to direct search. A learning system which discovers evaluation functions uses various techniques to generate an evaluation function from a solution that has been found. While such techniques are useful in domains where numeric evaluation functions are appropriate, other methods must be used to acquire heuristics that can only be stated in symbolic terms. One technique for learning symbolic conditions begins with very specific rules and generalizes as more information is gathered (Winston, 1984). An alternative approach starts with an overly general rule and generates more specific versions through a process of discrimination (Langley, 1982). Yet another approach incorporates aspects of both the generalization and discrimination methods (Mitchell, 1977).

5.2.2 Generalization and Discrimination Learning

Two types of techniques have been dominating research on condition finding: generalization and discrimination. In *generalization* learning, the learning system initializes the first positive instance as the initial rule. Here the rule represents a hypothesis of the concept to be learned. When a new positive instance is encountered, the learning system compares the current hypothesis with the new positive instance and finds the common features shared by the two structures. Then the hypothesis is redefined in terms of this common structure. In the case of the complex representation being used for the description of examples, it is possible to have more than one set of common structure and thereby have more than one generalization. Possible generalizations constitutes a rule space. Given the rule space, the learning system must carry out some type of search through the space of possible rules; a depth-first search (Winston, 1984) or a breadth-first search (Hayes-Roth and McDermott, 1976). The search can be aided with heuristics which suggest the most probable generalization out of possible generalizations (Winston, 1970). The learning system also can search the instance space to select those most likely to provide new information. Therefore, a learning system can be viewed as performing search through dual spaces: an instance space and a rule space (Simon and Lea, 1990).

When a depth-first search is used to search through the rule space (Winston, 1984), the learning system needs to have the ability to backtrack through this space because there is no guarantee that the learning system always select the right generalization. The learning system also needs to retain positive instances of a concept as well as negative instances in order to be able to backtrack. In the case of a breadth-first search being used (Hayes-Roth and

McDermott, 1976), the learning system needs never backtrack and does not need to retain positive instances. However, negative instances must still be retained for the case when they are to be used in rejecting overly general rules.

Mitchell (1977) proposed a *version space* technique which has done away with the need for negative information as well. A version space is simply a way of representing the space of all concept descriptions consistent with the training instances seen so far. Since a version space keeps track of all versions that are permitted by the data so far, the learning system does not need to backtrack and also does not need to retain negative instances. His version space technique maintains a set of maximally general versions (MGVs) and a set of maximally specific versions (MSVs). The set of MGVs contains maximally general patterns for legal rules (or operators). The set of MSVs contains maximally specific patterns for the rules. For a legal rule R, a maximally general version may say that R applies at all times whereas a maximally specific version may say that R applies only at a very particular situation. The truth lies in between. A particular algorithm, known as the candidate elimination algorithm, is used to manipulate the version space which is no more than a data structure. The algorithm begins by initializing the MGVs to the most general concept descriptions and the MSVs to the most specific concept descriptions available in the pattern-description language. As new positive instances are encountered, the specific boundary is made more general. As new negative instances of a rule are encountered, the general boundary is made more specific. Moving the specific boundary in the direction of greater generality can be considered as a breadth-first search from specific patterns to more general ones. The objective of this search is to compute a new boundary set which is sufficiently general so that it does not rule out a newly encountered positive

instance. Correspondingly, moving the general boundary in the direction of greater specificity can be considered as a breadth-first search from general patterns to more specific ones. The objective of this search is to compute a new boundary set which is just specific enough to rule out a newly encountered negative instance. Eventually the boundaries meet, and the process converges on one or more patterns that lie in the middle. Whether this pattern is actually correct depends on how well pattern-description language fits the domain.

The basic approach of generalization has a number of drawbacks that limit its value (Langley, 1987; Charniak and McDermott, 1985). First, because they examine features that are held in common between examples, generalization-based strategies do not lend themselves to the discovery of disjunctive concepts. When confronted with positive examples of disjunctive rules, these systems overgeneralize and cannot recover. Mitchell's (1977) version space approach is also subject to this drawback because it also finds features held in common by all positive instances. Second, generalization-based learning systems have difficulty handling noisy data. This also results from their dependence on finding commonalities in examples. If even one of these examples is faulty, then the entire learning sequence is thrown into confusion. Finally any program that learns through generalization would have serious difficulty responding to an environment in which the conditions predicting an event actually changed over time. If a tutor decides to modify the definition of a concept in the middle of a training session, a system that searched for common features would rapidly become very confused. It is argued that the ability to recover gradually from changes in its environment would be a definite advantage in real-world-settings. Langley (1983a, 1985,

1987) proposed discrimination learning as an alternative which has the potential to overcome these difficulties.

5.2.3 A General Theory of Discrimination Learning

Langley (1983a, 1985, 1987) proposed discrimination learning as a general learning mechanism. Discrimination learning is a type of empirical learning in that a learner attempts to learn from its past problem solving experience. Discrimination learning is a type of strategy learning in that learning occurs in the process by which general but weak methods are transformed into powerful, domain-specific search heuristics.

Discrimination learning starts with very general rules containing few conditions, and add more conditions as the need arises. Rather than looking for features held in common by all positive instances, these methods search for differences between positive and negative instances. These differences are then used to further specify the conditions under which a concept or rule should apply. The goal of discrimination learning is to compute more conservative versions of the production with additional conditions that will prevent their applications in the undesired case. Langley (1987) demonstrated the generality of discrimination learning by applying it to strategy learning tasks as well as concept learning tasks.

In concept learning tasks, Langley's program, called SAGE (Langley, 1983a), starts with an overly general rule which would predict that all examples are positive instances of a concept. Once an example is available, the program makes a prediction, and compares the prediction with the correct answer provided by a tutor. When the program correctly predicts a positive instance, it stores the instance for being used in the case of recovering from errors of

commission. An error of commission occur when the program predicts an example as a positive instance while it is a negative instance. There is another type of errors, called errors of omission. Suppose the program predicts it as a negative instance of a concept while the example is actually a positive instance of a concept, an error of omission has occurred. When an error of omission occur, the program designates a new very general rule that predicts any situation to be a positive instance.

When an error of commission has occurred, the program evokes the discrimination process in an effort to generate more conservative versions of the responsible rule with additional conditions that will prevent their application in the undesired case. The discrimination process retrieves the most recent positive instance of the rule that make a wrong prediction and compares it with the current negative instance with the goal of finding differences between the two. The former instance is called a selection context, and the latter is called a rejection context (Bundy et al., 1985; Langley, 1987). For each difference, the program creates a new rule which is identical with the old rule but with a difference as an additional condition. When an differential element is in a selection context and not in a rejection context, the element is just added to a new rule as a extra condition. On the other hand, when an differential element is in a rejection context and not in a selection context, the element is negated and then added to a new rule. The added condition ensures that the new rule would still be true when the original rule correctly applied, but would not be true in the undesired situation. Langley (1983a) further showed that the discrimination mechanism also gives the program the capability of learning disjunctive concepts, which causes a trouble in generalization methods. It is argued that learning disjunctive concepts is possible in the discrimination

technique because the program focuses on a single positive and a single negative instance at a time. Langley also argued that the strengthening process, when it works together with the discrimination process, is particularly effective in handling noise, be it positive or negative. The strengthening process weakens or strengthens the measure of success attached to each rule whenever it make a correct prediction or incorrect prediction respectively.

An example may clarify the discussion. Suppose a concept to be learned is "large or blue" and the program initial is provided an overly general rule that has no condition. Presented a positive instance "large and blue", the program predict it correctly as a positive instance and stores the instance in the memory. Encountering a negative instance "small and blue", the program which has only an initial rule predict it incorrectly as a positive instance and an error of commission has occurred. Given the error of commission, the program evokes the discrimination mechanism which retrieve the most recent positive instance, that is "large and blue" and compares it with the current negative instance "small and blue" to find one difference, "large" which is in the selection context but not in the rejection context. The program then creates a rule with the condition (large). Given yet another negative instance "small and red", the program, which has now an initial rule and a rule with a condition (large), predicts the instance as a positive instance, because even by this time the initial rule may have the largest strengthening value. The discrimination process is evoked again and create a rule with a condition {red). Through these processes, a disjunctive concept can be learned and each disjunct is represented as a separate rule.

The discrimination mechanism applied to strategy learning tasks is in principle same as the one described in concept learning tasks, except a few

complications (Langley, 1985). First the program solves a given problem with legal rules by the depth-first or breadth-first search to find solution paths and alternative paths. Using several heuristics (Langley, 1985, p. 234-235), the program identifies desirable moves from undesirable moves . Comparing the good move to the bad move, the discrimination mechanism generates variants of rules. Since in strategy learning there can be a number of legal rules or operators with several variables, a selection and a rejection contexts keeps variable bindings as well as all elements in working memory at the time when variable bindings occur. Differences between two sets are generated by a path-finding process which starts from analogous symbols in the two sets of bindings and attempts to find some path through the good elements that has no analogous path through the bad elements. The path finding process also searches for paths through the bad elements that have no analogous path through the good elements. Based on the difference that is found, the program constructs the variant of the old rule. Like in concept learning, the discrimination process is showed to learn a disjunctive concept. When a problem's search space is so large that the program cannot continue its processes until solution paths are derived, the program may use a number of heuristics (Langley, 1985, p. 223-226), which allows the program to learn while doing.

5.3 Empirical Learning

Empirical learning has been the dominant paradigm in machine learning and therefore there are many models developed. Such models include the ACT* model (Anderson, 1976, 1983), the SOAR model (Laird, Rosenbloom, and Newell, 1986), and many others (Anzai and Simon, 1979; Holland et al., 1986;

Langley, 1983b; Neches,1987; VanLehn, 1987;1990). All these systems focused on procedure learning within the production systems framework (see Chapter 5 for production systems). Procedure learning addresses the question of how knowledge encoded declaratively in the memory can be transformed to procedures, that is rules, which are more efficient. In this perspective, learning is viewed as the piecewise acquisition of condition-action pairs.

Although empirical learning models are based on different hypotheses about learning mechanisms, they share two basic assumptions (Ohlsson and Rees, 1991). One is that when faced with a novel task and with insufficient knowledge to generate a solution, a learning system applies weak methods such as heuristic search to generate a task-relevant behavior. The other is that a learning system examines solution paths or traces of the problem-solving steps and creates new procedures by various ways such as generalization, discrimination, and composition (Anderson, 1983; Langley, 1984; VanLehn, 1989). The basic insight underlying most empirical learning models is that we can generate heuristically useful conditions for legal rules through syntactic manipulations on data, solution paths, without considering domain principles. Those conditions gained would reduce the problem space at the next application. Therefore, it can be said that learning indeed occurs because search efforts are substituted with knowledge.

Models of empirical learning are indeed quite successful in explaining human learning in many cases (Anderson, 1983; VanLehn, 1982; Langley, 1983a). Nonetheless, from the perspective of cognitive diagnosis empirical learning models are subject to a significant criticism that procedures are typically constructed in order to produce particular sequences of steps, with little attention to the meaning of those steps. This is because a procedure is

acquired in isolation from its conceptual basis. If it is the case that the goal of education is to enable students to derive or construct procedures from the concepts and principles of a domain of interest, it may be desirable to relate procedure construction with concepts and principles. The next section provides arguments supporting this view.

5.4 Importance of Domain Principles in Procedure Construction

This study is based on the belief that procedures construction is based on domain principles rather than syntactic manipulation on data. This belief certainly assumes that if students understood concepts and principles of a domain, then errors would disappear, procedures would be better retained, and would transfer more easily to novel problem contexts (Ohlsson and Rees, 1991). Some evidences about conceptually based procedure construction in counting are reviewed in earlier chapters of this study (Gelman and Meck, 1983, 1986; Resnick, 1982, 1983, 1984). It is also argued that procedures learned with understanding increase the likelihood that an appropriate procedure will be recalled and used effectively (Hiebert and Lefevre, 1986). The last assumption of the transfer of cognitive skills appears to be intuitively clear, nonetheless determining the breadth and depth of transfer of cognitive skills is not a simple matter. In fact, the study of transfer has a long history but yet there seems to be no convergence at all.

A fundamental question in the study of transfer is whether transfer is limited in scope or whether it is broad and ranges across diverse disciplines (Singley and Anderson, 1989). The broad view, first advocated as the doctrine of formal discipline, was challenged by associationists like Thorndike, who claimed that transfer was quite specific and was based on the existence of

identical elements, known as the theory of identical elements. The Thorndike's theory of identical elements, however, was challenged by the gestalters like Werthemer, who showed that a broad range of transfer outcomes was possible and that the generality and applicability of knowledge was largely dependent on its representation-functional relations among elements-rather than identical elements. To gestalt psychologists, transfer occurred not through the piecemeal sharing of common elements but rather through the transposition of an entire structure. Based on the degree of transfer, gestalters then differentiated senseless learning or rote learning from meaningful learning. Senseless learning is the kind which would show little or no transfer whereas meaningful learning would show quite a bit .

With the advent of the information processing psychology, many traditional learning and transfer issues were temporarily set aside (Newell and Simon, 1972). One topic that has received considerable recent attention, however, is analogical transfer in problem solving (Anderson and Thompson, 1989; Gentner, 1989; Holyoak and Thagard, 1989). Analogical problem solving starts with reminding of a similar problem (source) whose solution is known. Then the solution of that problem is mapped to the current problem (target). To test this kind of transfer, many studies use isomorph-problems that differ in terms of superficial features but have the same problem-solving operators and search space. Hayes and Simon (1977) found that subjects in most cases failed to notice similarities between problems and drawing on analogous solutions. Gick and Holyoak (1980, 1983) reported similar findings in their subsequent studies. Subjects often had difficulties in drawing correspondences between the two domains and consequently in solving the target problem even when the were informed of the relevance of the source (Hayes and Simon,

1977; Kotovsky et al., 1985). The difficulty in many cases seems to be that subjects adopt representations of the two problems that rely too heavily on superficial features, which are radically different, but not heavily enough on deep, functional relationships, which are same. Expert-novice studies provides some evidences supporting this argument.

Expert-novice studies generally suggest that novices typical use shallow representations which mostly rely on superficial features whereas experts exploit deep representations which capture functional relationships (Chi et al., 1981; Chi et al., 1982). Chi et al. (1981) asked novices and experts to sort physics textbook problems on any basis they wished. Novices did so on the basis of kinds of apparatus involved (lever, inclined plane, balance beam, and the like), or the visual features of the diagram accompanying the problem. Experts, on the other hand, classified the same problems on the basis of the underlying physics principle that was needed to solve the problem (e.g., energy laws, Newton's second law). Clearly, novices depended on superficial features of problems for categorization, while experts sorted on the basis of abstract, solution-relevant features. Novices, hence, would have more trouble transferring solution-relevant features of the source to the target problem solving than experts. Indeed, Smith (1986) showed that substantial transfer was possible between isomorphs, given sufficient practice on source problems.

What can be understood from the above discussion is that transfer is the key determinant of meaningful learning. And also that meaningful learning is possible only when subjects have deep representations. Deep representations contain domain principles. The usefulness of deep representations lies in utility of aiding the interpretation of the problems. Deep representations are, therefore, the characteristics of experts' representations. Thus it can be argued

that any learning model which aims for diagnosis and tutoring must construct procedures similar to the form and content of a deep representation. Empirical learning is based on purely syntactic notion (Michalski, 1983). Empirical learning systems, because of their basic assumption of syntactic manipulation on data, are less likely to produce such representations than rational learning systems in which learning occurs with the help of domain principles. There is no guarantee that such a learning will be useful or even semantically well-formed when interpreted in the real world (Dejong, 1988).

5.5 Rational Learning

Unlike empirical learning models which are profound in machine learning literature, there are quite few rational learning models. Two prominent models are explanation-based learning and the state constraint. Explain-based learning uses domain principles to justify generalizations . All empirical learning models must make an unjustified inductive leap when it is assumed that a regularity that has been true in the past will be true in the future. Explain-based learning avoids this drawback by using domain principle to explain why the regularity will hold. Although explain-based learning is a good alternative for this study, it is decided to choose the state constraint theory for this study.

Ohlsson and Rees (1991) points out some technical differences between explanation-based learning and the state constraint theory. Explain-based learning systems employ their principled knowledge in the construction of an explanation. As a result, explanation-based learning systems require domain theories complete enough so that the required explanation can be constructed. In the state constraint theory, on the other hand, principled knowledge is used to evaluate search states. It is argued that Ohlsson and Rees's system can

operate with partial or incomplete domain theories. This point is viewed important, given the fact that human can perform with even incomplete knowledge although they make errors.

The state constraint theory of Ohlsson and Rees (1991) is the extension of Langley's (1983a) discrimination learning. The system proposed by Ohlsson and Rees, called Heuristic Searcher (HS), starts with very general rules and add more conditions as learning occurs. In HS, learning is viewed as the cognitive function of conceptual understanding. The state constraint theory views that performance with understanding occurs when the problem solver monitors his performance on the problem by comparing the successive states of the problem with what he knows about the task environment, i.e., principles about the environment. Learning occurs when an incorrect or incomplete procedure generates a problem state that is inconsistent with the principles. In the state constraint theory, therefore, a set of principles constitutes conceptual understanding, which are compared with the successive states of the problem, and principle violations trigger learning. Ohlsson and Rees developed hypotheses in detail in terms of understanding, performance, and learning (see Figure 5.3).

Figure 5.3. Hypotheses of the state constraint theory

Hypotheses about understanding

1. Understanding consists of knowledge about the task environment.
2. Knowledge is declarative rather than procedural.
3. Understanding consists of principled rather than factual knowledge.
4. Principles constrain the possible states of affairs.

Hypotheses about performance

1. Thinking is heuristic search.
2. Principles constrain search through state evaluation.

Hypotheses about learning

1. Constraint violations trigger procedure revision.
2. Constraint violations inform procedure revisions.

Hypotheses about understanding basically specify that understanding consists of principled knowledge. Principled knowledge is represented declaratively and is used to constrain the possible states of affairs. These hypotheses about understanding in general concur with findings in the studies of cognitive development. Hypotheses of performance are extensions of the major hypothesis uphold in cognitive science, namely, that problem solving consists of heuristic search, carried out by a production system architecture. Hypotheses about learning answer two basic questions of learning: critic and modifier (Bundy et al. 1985). In these hypotheses, constraint violations identify when a revision is needed and also inform information of how the faulty procedure should be revised. Ohlsson and Rees (1991) proposed two algorithms which could be used for the revision of faulty rules.

HS is based on these hypotheses. Ohlsson and Rees (1991) apply HS to the domain of counting and of arithmetic, and find that HS is able to

successfully construct procedures from conceptual knowledge represented as state constraints.

Heuristic Searcher (HS) consists of procedural knowledge, conceptual or principled knowledge, and a learning mechanism. A procedure consists of a collection of rules. Principles are represented in HS as state constraints. A state constraint C is an ordered pair <Cr Cs> of patterns in which each pattern is similar to the condition of a production rule. The left-hand pattern Cr, is called the relevance pattern, because it determines the class of search states to which the constraint is relevant. The right-hand pattern Cs is called the satisfaction pattern, because it encodes the criterion a state must match i order to satisfy the constraint (given that the relevance pattern matches). The relevance and satisfaction patterns are matched against the search states with the same pattern matcher that matches the production rule conditions, so new computational machinery has to be postulated. A cycle of HS consists of selection of a state, matching productions, and matching constraints. A cycle begins by HS selecting an unexpanded search state as the current state. The system then matches all production rules in the current procedure against the selected state. The system then matches its constraints against each new state and records which constraints, if any, are violated by that state.

HS has two modes of operation: performance and learning. In performance mode, HS perform best-first search with the number of constraint violations as the cost function. The idea of using the number of constraint violations as the evaluation function is similar to the one proposed by Langley et al. (1990). The evaluation function implies that HS prefers solution paths that are more congruent with its principled knowledge over those that are less congruent. In learning mode, HS preforms a breadth-first search, because it

needs to stop as soon as a constraint violation occurs. Given a constraint violation, HS applies its learning mechanism to the faulty rule which generated the constraint violation and to revise it.

The state constraint theory of Ohlsson and Rees (1991) merits attention in several aspects. It uses a separate, novel representation for principled knowledge, called state constraint, and consequently opens a path to meaningful learning. A state constraint is a two-part pattern that a search state has to satisfy in order to be valid. A state constraint is not a production rule or an inference rule. It is not a Horn clause. A state constraint does not guarantee that its right-hand side is true when its left-hand side is true; it claims that the right-hand side ought to be true. HS's learning can be viewed as trading evaluation selectivity with generative selectivity. Search efforts are traded with powerful knowledge. Thus, HS is a strategy learning system but utilizes a rational learning mechanism. The rational learning mechanism embedded in HS constructs a target procedure by constraining an initial procedure until it is consistent with the relevant principles.

5.6 Summary

This chapter reviews diverse perspectives and methodologies of cognitive diagnosis. In this study, cognitive diagnosis is viewed as consisting of inferrring a student model and producing error descriptions based on a inferred student model. Error descriptions can be used in an instruction module to decide the best remediation instruction.

Three approaches of describing errors are reviewed, and the reconstruction approach is taken for this study. Even in the reconstruction approach, there are two approaches. One approach reconstructs buggy

procedures on the basis of syntactic manipulations of symbols, while the other reconstructs faulty procedures on the basis of semantic rationalization. Since the constructivism assumption itself dictates semantic rationalization, this study attempts to model the process of semantic rationalization.

This study finds alternative methodologies for reconstruction: planning and learning. Learning is favored, and especially a rational learning mechanism called the state constraint theory (Ohlsson and Rees,1991), is chosen as a formalism for this study.

Chapter 6 A MODEL OF COGNITIVE DIAGNOSIS

In this chapter, a model of cognitive diagnosis, called PSCD (Production System for Cognitive Diagnosis), is proposed. PSCD is based on three constraints, and uses state constraints for expressing conceptual knowledge (Ohlsson and Rees, 1991) and productions for expressing procedural knowledge. The main hypothesis of PSCD is that conceptual understanding underlies procedural performance. In other words, PSCD hypothesizes that student errors arise from using buggy procedures that are constructed with incomplete understanding of domain principles.

PSCD is written in Common LISP. It has two modes of operation: performance and diagnosis. In the performance mode, PSCD solves a problem with given knowledge by searching the search space of the problem. In the diagnosis mode, PSCD diagnoses a student's inadequate understanding of a domain with the student's problem solving traces. Writing a program of a production system would have been a horrible experience without such valuable references as Anderson et al. (1987), Winston and Horn (1987), and Charniak and McDermott (1985). Most ideas employed in PSCD come from these references. Especially, the pattern matcher of PSCD is a modified one of Anderson et al. (1987, Chapter 15).

Section 6.1 presents central features of PSCD. Sections 6.2 and 6.3 explain representation languages of PSCD. Section 6.3 explains how PSCD detects errors and . In Section 6.4, PSCD is applied to a standard counting task, the same task as Ohlsson and Rees' (1991) HS is applied. In Section 6.5, PSCD is applied to a more complex task of transportation problem solving whose modeling is our main objective of the study. Section 6.6 summarizes this chapter.

6.1 The Central Architecture for Performance and Cognitive Diagnosis

There are five main characteristics associated with PSCD (Figure 6.1). These characteristics are similar to those of Newell's (1990) SOAR and Ohlsson and Rees' (1991) HS. The first of these states that search through problem spaces is the framework within which knowledge plays its role and in which task assessment is assessed. Two types of knowledge operate in the problem space. Conceptual knowledge guides the search and procedural knowledge implements the operators. Conceptual knowledge is represented as state constraints and procedural knowledge is represented as a set of productions. PSCD initially evokes the problem-space search with a set of productions. The problem-space search can consider one state at a time. At each state of the problem search, the knowledge search finds the relevant constraints and guides the problem-space search. At the performance mode, PSCD uses the number of constraints violation as the evaluation function to assess its progress in the problem-space search as in HS (Ohlsson and Rees, 1991). At the diagnosis mode, there is no need of problem-space search and of task assessment because at this diagnosis mode PSCD reads the trace of problem solving and only reports the constraints violated in the trace given to it.

Figure 6.1. The main characteristics of PSCD

1. Problem spaces represent all tasks
2. Productions provide procedural knowledge.
3. State constraints provide conceptual knowledge.
4. Attribute/object/value representation for all facts.
5. Goals direct all behavior.
6. The strengthening process adjusts the state of conceptual understanding.

In PSCD, all entities are represented as object-attribute-value (OAV) triples. OAV representation is used in SOAR, HS, and many others. Conditions, actions, and working memory elements all are OAV triples. Objects are defined by the set of attributes and values associated with them. Two objects are the same if they have the same attributes and values. OAV representation is argued to be easy to manipulate and to admit combinatorial variety. PSCD is a hierarchical production system like ACT* (Anderson, 1983). It does not keep a goal stack, rather goals are chained through productions.

One seemingly essential characteristic that does not appear in the list is about the learning mechanism. It has been argued that ITS must have a runnable learning mechanism as its part in order to build a student model. A student model built thereby can be used as a basis for assessing a student's weaknesses. In the proposed architecture, however, it is not necessary to assume a specific revision mechanism which creates new rules in response to credit assignment information (Ohlsson and Rees,1991; Langley,1985). It is viewed that student errors arise from using buggy procedures. From the perspective of constructivism, buggy rules are constructed due to missing constraints, i.e., incomplete understanding of domain principles. Thus it is possible to represent a student model as a proper subset of a complete set of constraints, and to diagnose student errors as lacking one or more missing constraints.

PSCD, however, hypothesizes a revision mechanism, called strengthening process. The strengthening process is used in many learning systems (e.g., Langley and Ohlsson, 1982; Anderson, 1983) and is advocated as a superior means for modeling errors (Reason, 1990). ACM (Langley and Ohlsson, 1982) and ACT* (Anderson, 1983) associated strength values with

productions and revise values as learning occurred. In contrast, PSCD associates strength values with constraints.

A correct set of constraints can be viewed as all having strength values higher than a certain threshold value and hence all constraints are applicable for state evaluation. A perturbed set of constraints, on the other hand, refers to a set of constraints some of which have strength values higher than a threshold value and the others do not. Those constraints with strength values lower than a threshold value are not applicable for state evaluation and therefore allow incorrect states to be considered, consequently generating errors. This study does not focus on such issues as what value a threshold be and how the strength process works. Rather it is posited that one can build such a mechanism and determine an appropriate threshold value later, when a fully implemented ITS is available and makes it possible to collect a large amount of data, i.e., students' problem-solving traces. In this study, it is simply assumed that a constraint is either applicable or not applicable. The question becomes whether PSCD is able to correctly identify errors from the problem-solving trace, and to suggest violated principles associated with those errors.

6.2 Representation of PSCD

Determining an appropriate representation is not an easy task at all. VanLehn (1983, p. 249) observed that "*Getting the knowledge representation language "right" allows the model to be made simple and to obey strong principles*" (italics in original). VanLehn (1983) admitted that Repair theory's representation language had undergone five major changes and each change had profound effects on the simplicity and principledness of the model. This study also have undergone the similar experience. Several representational languages which looked promising are tried. These included, although not

limited to these, display-based representation (Larkin et al., 1980; Larkin, 1981; Larkin, 1982; Larkin, 1983; Larkin et al., 1984; Larkin and Simon, 1987; Larkin, 1989), OPS5 (Brownston et al., 1985), and ACT* (Anderson, 1976, 1983). It appears that each approach has unique advantages and hidden costs as well, and thereby limits the set of possible languages that can be used to represent the domain under study. The following subsections discuss representation for facts, for principled knowledge, and for procedural knowledge.

6.2.1 Representation for Facts

PSCD uses attribute-object-value (AOV) triples to represent facts. The first element of this structure is an attribute of an object in the problem domain. For example, one may wish to describe the status of a line in a transportation tableau whether it is deleted or not. The name of the line and the value of the attribute follows. Thus one may represent a line called r1 which has been deleted as (status r1 deleted). Although it looks as simple as this, there are several important aspects to be considered.

Charniak and McDermott (1985, pp. 11-14) proposed three principles of representation (Figure 6.2). The first principle is concerned with the situation in which a referent in a representational language is not clear. For example, when we talk about "Jack", there can be more than one interpretation of "Jack" because there can be several persons whose names are all "Jack." This problem is called *referential ambiguity*. The solution is to make up names for each individual so that one name uniquely corresponds to one individual. Such unique names are instances or tokens of the class "Jack." The second principle is concerned with ambiguity in predicates. The problem of ambiguity in predicates is called *word-sense ambiguity*. For example, representation for a

sentence like "Jack caught a ball" may come out like (jack-2 caught ball-17). The problem of this representation is that the meaning of "caught" is not clear because the word "caught" in English has more that one meaning. As before the solution is to make up names for each word senses so that one name uniquely correspond to one word sense. Thus it would look like (jack-2 caught-object ball-17). The last principle requires one to clearly indicate *functional structure*. In the previous example, the functional structure of the sentence distinguishes the "catcher" (jack-2) from the "caught" (ball-17). The solution is to use the order of symbols there in exactly the same way that English does. Thus (jack-2 caught-object ball-17) is not the same as (ball-17 caught-object jack-2). According to Charniak and McDermott, for historical reasons representations are typically written with the predicate first. Thus we would write the previous example as (catch-object jack-2 ball-12).

Figure 6.2. Principles of representation

1. Representation must make the choice of referent explicit
2. All predicates in an representation must be unambiguous.
3. Representation must clearly indicate functional structure. English uses word order to express what we call functional structure.

AOV triples naturally captures the functional structure by specifying the structure with an attribute first, an object second, and a value as the last. Another key idea that may be useful in developing AOV triples is the distinction between ISA relationships and INST relationships. INST says that a particular individual is a member of some class. (element). ISA says that one class is a more general version of another.(subset). For example, suppose there is an

elephant whose name is "Clyde". We may say that "Clyde" is an INST (instance) of the class "elephant" and the class "elephant" ISA higher animal.

INST and ISA are two general relationships between objects. Some arbitrary relationships between objects can be hypothesized in addition to INST and ISA. These arbitrary relationships are more likely domain-specific. Sections 6.4 and 6.5 discuss domain-specific relationships hypothesized in the specific domain chosen for application.

6.2.2 Representation for Principled Knowledge

Principled knowledge refers to a collection of general principles about the task environment. Principled knowledge constitutes a problem solver's conceptual understanding of a domain. In the state constraint formalism, principles are stated in declarative forms.

A state constraint C is an ordered pair <Cr Cs> of patterns in which each pattern is similar to the condition of a production rule. The left-hand pattern Cr is called the relevance pattern, because it determines the class of search states to which the constraint is relevant. The right-hand pattern Cs is called the satisfaction pattern, because it encodes the criterion a state must match in order to satisfy the constraint, given that the relevant pattern matches.

For example, consider the domain principle for counting. A task of counting requires one to count a set of unordered objects. A representation for counting consists of symbols for objects (x1, x2, ...), numbers (n1, n2, n3,...), and sets, and four properties (first, current, answer, and origin) and four binary relations (next, associate, member, and after) among objects. To count a set of unordered objects is to select repeatedly an object from that set, to increment the current number, and to associate the new number with the selected object.

When all objects in the set have been associated with numbers, the last number to be associated with an object is asserted to be the answer to the counting problem. One counting principle proposed by Gelman and Gallistel (1978) is the one-to-one mapping principle which states that counting consists of establishing a one-to-one mapping between numbers and objects. Ohlsson (1991) breaks this principle down into four components and represents them as state constraints (see Figure 6.5). For each constraint, the relevant pattern Cr is shown to the left and the satisfaction pattern Cs to the right, separated by the arbitrarily chosen symbol "***".

For example, let us take a look on the first principle of standard counting in Figure 6.5. The first constraint in the one-to-one mapping principle denotes that if the object X1 is associated with the number N1 and also with the number N2, then the number N1 and the number N2 must be equal. Similarly, the second constraint states that if the object X1 is current and X1 is after X2, then X2 must be associated with a number N. From this example, it may become clear that the relevance patterns determine the applicability of the constraint and satisfaction patterns test for its success.

6.2.3 Representation for Procedural Knowledge

PSCD represents procedural knowledge in a set of productions. A production is an ordered pair of conditions and actions. Viewing from the perspective of search, a production or a set of productions represent an operator which transform a state to other state. The appropriate actions to these productions are determined rather easily because we can observe analogous actions in the environment. However, conditions which specify the appropriate context for actions to be applied are seldom obvious (Langley, 1985). As seen

in Chapter 5, studies in strategy learning systems focus on deriving such conditions from examining a lot of examples or data. Neches (1981) suggests three types of conditions that a production may have: (a) conditions that define the context in which the production is applicable; (b) conditions that enable processing on the action side of a rule; (c) conditions that serve to affect flow of control

The first type of condition contain knowledge about the circumstances in which it is meaningful to execute the production. Typically this type of conditions are instantiated as goals. The second type of condition is usually necessary in order to assigning values to variables referenced on the action side. These are legal conditions which must be bound to some values in order to to instantiate actions. The third type of condition holds heuristic knowledge about desired sequencing of productions. Langley (1987) suggests the third type of conditions can be learned by a learning mechanism such as discrimination learning. Langley's discrimination learning (1987) is a type of empirical learning because it learns from experience. Ohlsson' state constraint theory (1991) is similar to Langley's discrimination learning (1987) in that it generates more specialized new rules during the course of learning, but it is a rational learning mechanism because it relies on conceptual knowledge in building new rules.

In PSCD, procedural knowledge consists of initial knowledge runnable to generate solutions. Productions representing procedural knowledge impose minimal guidance on the application of the operators. Two types of conditions, goal and legal conditions, are attached initially to productions. Goal conditions specify the meaningful context of productions. Legal conditions are concerned with retrieving bindings from working memory to instantiate actions; this allows

the actions to be executed. But the initial rules do not specify when, under what circumstances, or in what order the operators should be executed. Therefore initial productions impose a minimal assumption on the background knowledge of students. Students are expected at least to know what actions are available and their associated minimal conditions.

6.2.4 The Strengthening Process

The strengthening process adjusts the level of strength associated with a state constraint depending on its success or failure in explaining a problem-solving trace of a student. The strength of a state constraint represents the degree of understanding associated with a state constraint. Thus, for example, a state constraint with a higher strength than a threshold value may be considered in making a student model whereas a state constraint with a lower strength may not be considered. A set of state constraints with higher strengths than a threshold value and a set of productions together would make a runnable student model. Such a runnable student model can be used to predict student's errors or problem-solving traces in a different set of problems from those on which a student model is based.

There can be many schemes for the strengthening process. Anderson (1983), for example, associates the strength of 1 with a production when it is created, and increases its value when it is successfully applied and decreases its value by 25% when it applies and receives negative feedback. It is possible to devise a similar scheme of the Anderson's strengthening process in PSCD. PSCD, however, assumes the crudest strengthening scheme that hypothesizes only existence or non-existence of constraints in a student model. Thus, constraints can have either one of two values: zero (non-existence) or one

(existence). Examining problem-solving traces, PSCD will reports violated constraints as they arise. The set of reported constraints and their frequencies constitutes a basis for determining the best remedial instruction for correcting student's cognitive errors. It is assumed that this information would suffice to do that. Since it is the tutoring model that concerns the issue of determining the best remedial instruction, more definite conclusion could be achieved by the future study which includes the tutoring model as well as the diagnosis model. Until then, this study simply accepts the assumption.

This assumption implies two concerns. First, since this study emphasizes domain principles in explaining errors, explanations of why errors occur, i.e., student models, must be tied with domain principles. Second, given the first concern, information about violated constraints and, if ever needed by the tutoring model, their associated violated frequencies would be sufficient to determine the best remediation. In this study, it is assumed that PSCD augmented with a strengthening mechanism would suffice to generate explanations elegant enough to allow sophisticated remediations.

The strengthening mechanism of PSCD can be viewed as a perturbation technique that generates student models from expert models, because in the strengthening mechanism a student model is hypothesized as a subset of an expert model. The perturbation technique of PSCD is different from previous uses in that it operates on state constraints rather than on procedural productions (Brown and VanLehn, 1980; Young and O'Shea, 1981) or on preference productions (VanLehn et al., 1989) . Since the correct set of state constraints represents the conceptual understanding of an expert, the perturbed set of state constraints represents inadequate conceptual understanding or misconceptions on the part of novices.

6.3 Performance and Diagnosis of PSCD

Two major functions of PSCD are performance and diagnosis. In the performance mode, PSCD executes a procedure with a set of constraints. PSCD performs a best-first search with the number of constraint violations as the evaluation function. The top level function that starts the performance mode of PSCD is shown in Figure 6.3 The algorithm for a best-first search comes from Winston (1984, p. 98) and Winston and Horn (1989, p. 281). Initially a one-element queue consisting of the root node is formed. If there is (halt) in the working memory, PSCD stops. Otherwise, PSCD proceeds to select the best path in the queue. The best path is determined by first identifying an equivalent class of paths which all has the least number of constraint violations, if there are multiple paths in the equivalent class find another equivalent class of paths, which all have the longest path, from the least constraint violation class. If there are still multiple paths in the longest-path class, PSCD will select any path randomly.

Figure 6.3. The top-level function of performance

```
While (the queue is not empty)
  Begin
    If (there is halt) Then Stop
    Else
      Begin
        (Select the best path in the queue)
        (Extend the best path by applying productions)
        (Add extended paths to the queue)
        (Sort the entire queue by the number of constraint violations)
      End
  End.
```

The next step is to extend the selected best path to generate new possible states. PSCD finds applicable productions and all instantiations for each production. Each instantiation forms a new path. Each instantiation is fired and the resulting new state is checked to determine the nature and number of constraint violation. This information is attached to the new path. All new paths are generated through this process and added to the queue. The entire queue is then sorted according to the number of constraint violation, in the order of the least violation path first. The entire process continues until the goal is achieved (i.e., the "halt" production is fired) or there is no path in the queue.

In the diagnosis mode, PSCD makes a diagnosis in response to a problem solving trace. The process of the diagnosis mode is much simpler than that of the performance mode. The top-level function of PSCD for diagnosis is shown in Figure 6.4. An action sequence, often called action protocol (Kowalski and VanLehn, 1988; VanLehn et al., 1989b), is an input to the diagnosis component of PSCD. Diagnosis starts with selecting the first action in the action sequence. The action is tested if the action is one of the possible actions in the current state. Since all productions are overly general, once applicable a production would generate multiple instantiations. The action must be one of them. PSCD must hold this requirement because state constraints are very much dependent on productions in their developments and operations. When the action is not one of possible actions, PSCD may not be able to give correct diagnosis.

Figure 6.4. The top-level function of diagnosis

```
While (the action sequence is not empty)
  Begin
    (take the first action in the action sequence)
    If (the action is not a member of the set of possible instantiations of the
       associated production)
       Then Stop
   Else (find all constraints violated in the state resulting from applying the
        action)
  End.
```

The next step of diagnosis is to apply all constraints to the state resulting from firing the action, and find all relevant constraints. All relevant constraints that are found are tested if they are satisfied. Unsatisfied constraints, that are violated constraints, are reported. The tested action is popped our from the action sequence with which the whole process restarts again. This process continues until there is no action available in the action sequence.

This study is mainly concerned with the diagnosis of *principled* errors. Principled errors are those errors that can be seen as violation of one or more of constraints. This concern is based on the belief that most errors that can be found are rooted in inadequate understanding of domain principles or in failure of applying those basic principles to problems. This belief is based on the conclusions of a number of studies in cognitive development and cognitive science (Resnick and Omanson, 1987; Resnick 1982; Gelman and Meck, 1986; Chi et al., 1982; Chi et al., 1982; Dejong, 1988). It is argued that if students understood concepts and principles of a domain, then errors would disappear, procedures would be better retained, and would transfer more easily to novel problem contexts. Teaching domain principles, however, may not guarantee skilled performance. As Resnick (1982) observed in the case of arithmetic

problems, children frequently made errors even although they possessed enough principled understanding necessary to answer correctly. Resnick (1982) attributed this to an inadequate linking of the principled understanding with the syntax of the written algorithms. In other words, children have adequate understanding of principles relevant to the domain but they are not pretty sure when those principles are relevant. The state constraint formalism is a wonderful idea because it allows to represent principles in terms of when they are relevant and how they can be satisfied. Diagnosis of errors in PSCD is based on this idea.

As explained, diagnosis of principled errors is based on principled and procedural knowledge. One requirement for diagnosis is that actions observed in empirical data must be generatable from PSCD's productions. When this requirement is not met, PSCD cannot guarantee correctness of its diagnosis. When this is not the case, PSCD proceeds to find violated constraints for each action and report them. A violated constraint can be interpreted as incomplete understanding of the constraint on the part of the student. Incomplete understanding of a principle can be expressed having a lower strength value than a threshold value. The diagnosis system relates a user's incorrect performance to violations of constraints which are viewed as incomplete understanding of those constraints. Thus PSCD hypothesizes that incorrect performance can be modeled with a perturbed set of constraints and initial productions.

6.4 Modeling Standard Counting

With PSCD this study models standard counting first to validate its performance ability and to compare it with HS (Ohlsson and Rees, 1991). This

study takes standard counting as the first application because HS is applied to it and also because the strongest empirical evidence for the hypothesis that conceptual understanding facilitates procedure acquisition comes from the domain of counting. Our first application in standard counting shows how our program performs on the basis of a state constraint representation of the principles of counting. This study adopts from Ohlsson and Rees (1991) initial procedural knowledge (i.e., the problem space) and the model's principled knowledge (i.e., the state constraints that encode the counting principles). With these initial procedural and principled knowledge, it will be shown how the program performs a counting task.

Standard counting involves in counting a set of unordered objects. Greeno et al. (1984) coined the term "standard counting" to denote a standard procedure of counting. Its procedure is shown in Figure 6.5. Figure 6.6 shows a representation for standard counting. The representation includes three types of entities, their associated properties, and relations among entities. The objects represent physical objects such as dice (?). The set ToCountSet represents the collection of objects which need to be counted. Any object can be a member of the set ToCountSet, which means that the object need to be counted. An object that is not the member of ToCountSet is physically present but not need to be counted. Numbers are assumed to be retrieved from a number sequence. The only relationship between numbers is "next". For example, (next 3 2) means that the number "3" is next to the number "2." This representation, therefore, does not assume any prior knowledge of other relationships such as greater-than or less-than. When a number (n) is assigned to an object (x), a new assertion (associate x n) is created to represent the new relationship between the number and the object. The predicate "after" denotes

that the object x1 is considered after the object x2. Therefore, the "after" predicate represents the object sequence of consideration in problem solving.

Figure 6.5. A standard procedure of counting

1. Select repeatedly an object from the set required to be counted, increment the current number, and to associate the new number with the selected object.

2. When all objects in the set have been associated with numbers, the last number to be associated with an object is asserted to be the answer to the counting problem.

Figure 6.6. A language for standard counting

Types of entities
- (a) objects: x1,x2,...
  properties: first, current
- (b) numbers: n1,n2,.....
  properties: first, current, answer, origin
- (c) sets: ToCountSet

Relations
- (a) (next n1 n2): The number n2 is the successor of the number n1 in the number line.
- (b) (associate x n): The object x is associated with the number n
- (c) (member x ToCountSet): The object x is a member of a set ToCountSet
- (d) (after x1 x2): the object x1 is after the object x2 in a sequence.

Figure 6.7. A problem space for standard counting

The initial knowledge state

The initial knowledge state for standard counting contains the number sequence in which numbers are liked to its successor with the next relation, the set of objects to be counted (ToCountSet), and some additional objects that are not members of the ToCountSet

Operators

1. (PickFirst x)
   add: (first x)(current x)
2. (PickNext x1 x2)
   add: (current x2)(after x2 x1)
   delete: (current x1)
3. (Initialize n)
   add: (first n)(current n)
4. (increment n1 n2)
   add: (current n2)(after n2 n1)
   delete: (current n1)
5. (associate x n)
   add: (associate x n)
6. (assert n)
   add: (answer n)

Goal state

The goal is to reach a state in which some number has the property of being the answer.

Figure 6.7 shows a problem space for standard counting that builds on that representation. The problem space includes six operators. The initial knowledge state consists of a set of assertions representing number facts and objects which belong to ToCountSet as well as objects which do not. Applying operators to the initial knowledge state generates a set of new states. It is the search mechanism that navigates the problem space to reach a goal state. In a standard counting task, the goal state is a state which has an assertion (answer n).

Figure 6.8 shows working memory elements and initial rules for standard counting. Initial working memory elements represent the initial knowledge state for a counting task with 5 numbers and four objects. Each number is declared as an instance of the 'number' class. The number 1 has a property of being 'origin'. The number 2 has a relationship of being 'next' to the number 1. Similarly, the number 3 has a relationship of being 'next' to the number 2. Four objects are declared as instances of the 'object' class. Among them, three objects (b1, b2, b3) have relations of being member to the ToCountSet The ToCountSet is declared as an instance of the 'set' class and represents the set of to-be-counted objects.

The initial rules impose minimal guidance on the application of the operators. Their main effect is to retrieve bindings for the operator arguments from working memory; this allows the operators to be executed. But the initial rules do not specify when, under what circumstances, or in what order the operators should be executed. This means that PSCD initially does not know how to count. The set of initial rules represent an executable procedure, and its execution will generate task-relevant but incorrect behavior. Figure 6.7 and Figure 6.8 together represent the initial procedural knowledge of PSCD in this application.

Figure 6.8. Initial working memory elements and rules

Working-memory
(number 1)(origin 1)
(number 2)(next 2 1)
(number 3)(next 3 2)
(number 4)(next 4 3)
(number 5)(next 5 4)

(object b1)(member b1 ToCountSet)
(object b2)(member b2 ToCountSet)
(object b3)(member b3 ToCountSet)
(object b4)

(set ToCountSet)

Rules
1. (object x) ==> (PickFirst x)
2. (object x1)(current x1)(object x2) ==> (PickNext x1 x2)
3. (number n) ==> (Initialize n)
4. (number n1)(current n1)(number n2) ==> (increment n1 n2)
5. (number n)(current n)(object x)(current x) ==> (associate x n)
6. (number n)(current n) ==> (assert n)
7. (answer n) ==> (halt)

In order to generate correct behavior, PSCD needs principled knowledge. Building on counting principles, suggested by Gelman and Gallistel (1978), Ohlsson and Rees (1991) proposed five counting principles and represented them as state constraints as shown in Figure 6.9. The *one-to-one mapping principle* states that counting consists of establishing a one-to-one mapping between numbers and objects. They break this principle down into four component ideas: that an object is associated with at least one number, that an object is associated with at most one number, that a number is associated with at most one object, and that a number is associated with at least one object. The *cardinal principle* states that the answer to a counting problem is the last number to be associated with an object. This principle is broken

down into three component ideas: that the size of a set cannot be known until all objects in the set have been associated with numbers, that the answer is a number associated with some object, and that the answer is the last number considered.

The regular traversal principle states that traversing of the number line begins at unity and then follows the next relations. The state constraint representation breaks this principle down into four component ideas: that counting begins with the origin of the number line, that each number considered is the successor of the previous number, the numbers are considered one at a time, and that each number is associated with some object. The *order imposition principle* imposes a linear ordering on the objects counted. It is broken down into six component ideas: Only one object is designated as the first object in the ordering; objects are considered one at a time; no object is considered twice; an object is not considered after itself; the first object is never considered again; and finally, no object that is not a member of the to-be-counted set is considered. Finally, the actions of traversing the number line in the right way and imposing an order on the objects are not sufficient to produce correct counting. The two processes must be connected with each other in the right way. The *coordination principle* states that objects and numbers are associated with each other in the order in which they are attended to.

The state constraints in Figure 6.9 represent the principled knowledge of the PSCD system in counting application. Ohlsson and Rees (1991) claim that the set of state constraints is complete and is sufficient to determine correct counting. With the initial knowledge of the system being defined, PSCD is presented with an example counting problem shown in Figure 6.10. PSCD can correctly solve the problem.

Figure 6.9. State constraints

A. The one-to-one mapping principle
1. All object should be associated with at most one number.
(object x1)(number n1)(number n2)(associate x1 n1)(associate x1 n2)**(equal n1 n2)
2. Every object considered during counting should be associated with some number
(object x1)(object x2)(current x1)(after x1 x2)**(number n)(associate n)
3. A number should be associated with at most one object.
(object x1)(number n1)(object x2)(number n2)(associate x1 n1)(associate x2 n2)**(equal x1 x2)
4. For every number retrieved during counting, there should be some object with which it can be associated.
(object x1)(number n)(current n)(associate x1 n)**(current x2)

B. The cardinal principle
1. A number is the answer to a counting problem only if there are no objects that are members of the to-be-counted set but that has not been associated with some number.
(number n1)(answer n1)**(object x)(not (member x ToCountSet)(not (associate x n2)))
2. The answer to a counting problem is one of the numbers associated with some object.
(number n)(answer n)**(associate x n)
3. The answer to the counting problem is the last number to be considered in the counting process.
(number n)(answer n)**(current n)

C. The regular traversal principle
1. Initialize counting at the first number in the number line.
(number n)(first n)**(origin n)
2. Consider one number at a time.
(number n1)(number n2)(current n1)(current n2)(equal n2 n1)
3. The numbers should be considered in the order defined by the next relations.
(number n1)(number n2)(current n1)(after n1 n2)(not (equal n1 n2))**(next n2 n1)
4. For each number considered, the preceding number should be associated with some object.
(number n1)(Number n2)(next n2 n1)**(object x)(associate x n2)

D. The order imposition principle
1. Initialize counting with a single object.
(object x1)(object x2)(first x1)(first x2)**(equal x1 x2)
2. Do not consider an object that is already associated with a number.
(object x)(current x)(number n)(not (current n))**(not (associate x n))
3. Do not cycle back to the first object.
(object x1)(first x1)**(not (after x1 x2))
4. Do not consider an object after itself.

Figure 6.9. (Continued)

(object x1)(object x2)(after x1 x2)**(not (equal x1 x2))
5. Consider only one object at a time.
(current x1 x2)**(equal x1 x2)
6. Do not consider object that are not in the set of to-be-counted objects.
(object x)(current x)**(member x ToCountSet)
E. The coordination principle
1. Numbers and objects are associated with each other in the order in which they are considered.
(object x)(number n1)(number n2)(current x)(current n1)(associate x n2)**(equal n1 n2)

Figure 6.10. A program trace for an example problem

```
#S(INSTANCE RULE START ACTIONS NIL VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE PICKFIRST ACTIONS ((*ADD-DB (CURRENT B1)) (*ADD-DB (FIRST B1)))
    VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE INITIALIZE ACTIONS ((*ADD-DB (CURRENT 1)) (*ADD-DB (FIRST 1)))
    VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE ASSOCIATE ACTIONS ((*ADD-DB (ASSOCIATE B1 1))) VIOLATIONS NIL
    TOTAL 0)
#S(INSTANCE RULE PICKNEXT ACTIONS ((*DELETE-DB (CURRENT B1)) (*ADD-DB (AFTER
    B2 B1)) (*ADD-DB (CURRENT B2))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE INCREMENT ACTIONS ((*DELETE-DB (CURRENT 1)) (*ADD-DB (AFTER 2
    1)) (*ADD-DB (CURRENT 2))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE ASSOCIATE ACTIONS ((*ADD-DB (ASSOCIATE B2 2))) VIOLATIONS NIL
    TOTAL 0)
#S(INSTANCE RULE PICKNEXT ACTIONS ((*DELETE-DB (CURRENT B2)) (*ADD-DB (AFTER
    B3 B2)) (*ADD-DB (CURRENT B3))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE INCREMENT ACTIONS ((*DELETE-DB (CURRENT 2)) (*ADD-DB (AFTER 3
    2)) (*ADD-DB (CURRENT 3))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE ASSOCIATE ACTIONS ((*ADD-DB (ASSOCIATE B3 3))) VIOLATIONS NIL
    TOTAL 0)
#S(INSTANCE RULE ASSERT ACTIONS ((*ADD-DB (ANSWER 3))) VIOLATIONS NIL TOTAL 0)
```

6.5 Modeling Transportation Problem Solving

In this section, PSCD is applied to transportation problem solving, the main application of PSCD in this study. Transportation problem solving is usually taught in introductory. Management science has become an integral part of management education, because it provides a rich set of tools that help decision makers formulate and solve complex problems in a rational and systematic way. It has been repeatedly observed that use of management science tools improves the quality of decision-making.

Teaching and learning management science has not been easy. Many students have experienced difficulties in understanding the principles that underlie the formal methods of management science, and using principles in a specific problem solving contexts. In this study, transportation problem solving is chosen as a main application task because of its simple structure. In the following sections, first, transportation problem solving is described, and from this description, principled and procedural knowledge are extracted and represented with the representational language of PSCD. Then, PSCD is demonstrated to be sufficient to solve transportation problems.

6.5.1 Transportation Problem Solving

The transportation problem deals with the transportation of a product from a number of sources, with limited supplies, to a number of destination with specified demands, at the minimum total transportation cost (Lee et al., 1990). Transportation problems are a special type of linear programming applications. Although transportation problems can be solved by the simplex method, because of their special network structures, many other simple techniques have been developed. These techniques are generally more efficient than the

simplex method, but are actually extensions of the linear programming concepts and procedures.

The solution procedure for the transportation problem involves first finding an initial feasible solution and then proceeding iteratively to make improvements in the solution until an optimal solution is reached. Most textbooks present three methods for deriving an initial solution: northwest corner method, least-cost method, and Vogel's Approximation method (VAM). VAM is generally known to give a better initial solution than the northwest corner method or the least-cost method. This study applies PSCD architecture to VAM to study adequacy of the architecture in performing cognitive diagnosis.

VAM consists of making allocations in a manner that will minimize the penalty cost for selecting the wrong cell for an allocation. The general procedure for the VAM is shown in Figure 6.11 (Lee et al., 1990). From this description of the VAM procedure, it may not be hard to find actions needed for performing the procedure. Although there can be some different set of actions, this study employes the set of operators as shown in Figure 6. Operators are described in detail in Section 6.5.2. With these operators, a control structure of the VAM procedure is defined as in Figure 6.12. The control structure shows appropriate actions during the course of problem solving.

Figure 6.11. A general procedure of VAM

1. Calculate the penalty cost for each row and column. The penalty cost for each row is computed by subtracting the smallest cost in E_{ij} (denoting the cell in i-th row and j-th column) in the row from the next smallest cost E_{ij} value in the same row. Column penalty costs are obtained in the same way. These costs are the penalties for not selecting the minimum cell cost.

2. Select the row or column with the greatest penalty cost (breaking any ties arbitrarily). Allocates as much as possible to the cell E_{ij} with the minimum value in the selected row or column. As a result, the largest penalties are avoided.

3. Adjust the supply and demand requirements to reflect the allocation(s) already made. Eliminate any rows and columns in which supply and demand have been exhausted.

4. If all supply and demand requirements have not been satisfied, go to the first step and recalculate new penalty costs. If all row and column values have been satisfied, the initial solution has been obtained.

Figure 6.12. Control structure

```
While (there is an applicable rule) or (there is no halt) Do
  Begin
    if (all lines have zero amount)
      Then halt
    Else if (there is a line whose current value is zero)
          Then Delete-line
    Else If (top) and (there is a line with only one blank cell)
          Then Copy-allocation
    Else
     Begin
       While (every undeleted line is associated with an opportunity)
         Begin
           If (there is no calculated penalty)
             Then First-cp
             Else Next-cp
           Calc-penalty
         End
       Select-line
       Select-cell
       Determine-allocation-amount
       Adjust-amount
     End
End
```

From the above descriptions, this study identifies four principles underlying the procedure. These principles are shown in Figure 6.13. It may be the case that students who understand the above concepts well are able to derive an initial solution to a transportation problem without much difficulties. Their performance does not rely on rote procedures but meaningful procedures (Smith et al., 1989) because their performance rely on these principles. This study will demonstrate PSCD's capabilities with the example problem (see Figure 6.14) taken from Lee et al. (1990).

Figure 6.13. Four principles underlying the VAM procedure

1. The opportunity cost principle: We want to minimize the opportunity (i.e., regret or penalty) cost for selecting the wrong cell for an allocation. The motivation is to minimize the total allocation cost.

2. The least-cost allocation principle: We allocate to the cell with least cost in the selected row or column. The motivation is to minimize the total allocation cost.

3. Demand/supply requirements: We want to allocate all supply or demand in such a way that allocations do not violate requirements.

4. The order principle: This principle imposes an order in which problem solving proceeds. For instance, a line must be selected before selecting a cell.

Figure 6.14. An example problem (adopted from Lee et al.,1990)

From \ To	DEMAND 1	DEMAND 2	DEMAND 3	Supply
SUPPLY 1	8 70	5	6 50	120
SUPPLY 2	15	10 70	12 10	80
SUPPLY 3	3 80	9	10	80
Demand	150	70	60	280

6.5.2 Principled and Procedural Knowledge of Transportation Problem Solving

In this section, PSCD is applied to transportation problem solving. The application is discussed in the following order: a representational language for transportation problems (Figure 6.15), a problem space including a set of operators and initial knowledge state (Figure 6.16), working memory elements for an example problem and initial procedural rules (Figure 6.17), and lastly state constraints (Figure 6.18).

A language for transportation problem solving consists of three types of entities, their associated properties, and two types of relations (Figure 6.15). The entity 'line' designates a line in a transportation tableau and has properties of 'value', 'current-value', and 'status'. The value property and the current-value property of a line can be only a numerical value. The property status can take any one of values: nil, selected, or deleted. The value of the value property represents the amount to be allocated for a source or a destination. The value of the current-value property represents the allocable

Figure 6.15. A language for transportation problem solving

Types of entities

(a) lines: r1, r2, r3, ... c1, c2, c3.....
properties: value, current-value, status

(b) cells: e11, e12, e13,....
properties: parameter, value, status

(c) opportunities: op1, op2, op3, ...
properties: first, second, value

Relations

(a) (partof e11 r1): The cell e11 is part of the line r1.

(b) (relate op1 r1): The opportunity op1 is related to the line r1.

amount adjusted for the allocations which are already made. The status property contains information about whether the line is selected for an allocation or is deleted. the default value of the status property is nil, which means that any operation has not been performed on this line.

The entity 'cell' represents a cell in a transportation tableau and represents an allocation from a certain source to a destination. The cell entity has three properties: parameter, value, and status. The parameter property contains cost information, the value property represents an allocation amount assigned to that cell, and the status property represents whether that cell is selected, deleted, or nil (not deleted). The entity 'opportunity' keeps track of opportunities associated with lines. The opportunity has three properties: first, second, and value. The value of the property 'first' indicates the first cell selected for opportunity calculation. The value of the property 'second' indicates the second cell selected. The property 'value' contains the opportunity value calculated from two selected cells. The language for transportation problem solving includes two relations. The relation 'partof' specifies which cells belong to which lines. The relation 'relate' relates a line with an opportunity.

Figure 6.16 describes the initial knowledge state, operators, and the goal state which collectively determine a problem space for transportation problem solving. A transportation tableau is described with lines, cells, and their properties and relationships. There are nine operators operating in a problem space for transportation problem solving. The three of them, that are First-cp, Calc-penalty, and Next-cp, are operators that work for calculating penalties for lines. The operator Select-line selects a line, Select-cell selects a cell, Determine-allocation-amount determines an appropriate allocation amount,

and Adjust-amount adjusts current values of a row and of a column to reflect the allocated amount. Delete-line deletes a line. Copy-allocation copy a current value of a line and write the current value as an allocation for a blank cell. Operators are interchangeably called actions in the study because this set of operators are used to transcribe problem solving traces of students into action protocol (see Chapter 7). A operator has a set of arguments which are enclosed with parentheses. Arguments are prefixed with the symbol "=" to denote that they are variables. System related arguments which may not be easily comprehensible to readers are enclosed with brackets. For example, the operator First-cp has three arguments: =line, =op, and =op-list. It would be enough to understand the First-cp action with the information about the selected line =line. The rest arguments are system related arguments.

Figure 6.17 shows initial working memory elements for an example transportation problem and initial rules. The example problem has a tableau which three rows and three columns. Therefore there are nine cells in the problem. Each cell is defined as an instance of the 'cell' class. A cell is a part of a column line and a row line. For example, the cell e11 is a part of the column line c1 and also of the row line r1. The cell e11 has three properties: parameter, value, and status. The value of parameter property for the cell e11 is 8, the value of value property is zero, and the value of status parameter is nil which means not deleted.

Figure 6.18 describes state constraints in terms of four principles. The opportunity principle consists of fourteen constraints. They are (a) an opportunity value must be calculated for a line which is not deleted, (b) consider only one line at a time, (c) consider only one opportunity at a time, (d) do not relate more than one opportunity with a line, (e) do not relate more than one line

with an opportunity, (f) do not consider a line which is already associated with an opportunity, (g) a first cell of an opportunity must be part of a line which is associated with the opportunity, (h) a second cell of an opportunity must be part of a line which is associated with the opportunity, (i) a first cell of an opportunity must be a next-least cost cell in a line associated with the opportunity, (j) a second cell of an opportunity must be a least cost cell in a line associated with the opportunity, (k) A first cell of an opportunity must not be deleted, (l) a second cell of an opportunity must not be deleted, (m) select a line which has the largest opportunity value, and (n) select a line after all lines which are not deleted are associated with some opportunity. The principle of least cost allocations consists of two constraints: (a) select a cell which has a least cost and (b) select a larger current value of two lines of which a selected cell is part. The principle of demand-supply requirement consists of two constraints: (a) delete a line whose current value is zero and (b) a value of a line must be equal to a current value of the line plus total allocations made to the line. The order principle consists of five principles: (a) select a cell after a line is selected, (b) determine an allocation amount for a selected cell after the selection of line and cell, (c) adjust current values of lines given there is a determined amount, (d) delete a line with zero current value after adjustment, and (e) when all cells except one cell are deleted, do not calculate penalties.

Now the performance capability of PSCD are demonstrated with the example problem. The resulting output from PSCD is shown in Figure 6.19. A program trace in Figure 6.10 is the one randomly selected from the best possible paths that all can generate a correct solution. Each line that starts with the "#S" symbol represents a structured object showing information about a rule's name that are fired, instantiated actions, names of violated constraints,

and the total number of violations until that stage. For example, the second instance in Figure 6.19 shows that the rule FIRST-CP is fired with instantiated actions ((*ADD-DB (CONSIST-OF OPPORTUNITY (*PUSH G377 NIL))) (*DELETE-DB (CONSIST-OF OPPORTUNITY NIL)), there is no constraint violation from firing this instantiation, and the total number of violations until that state is zero. In the last instance, the total number of violations is zero. Hence the solution generated from this path is correct. This demonstrates that PSCD, with the defined set of constraints and rules, is sufficient to solve transportation problems.

Figure 6.16. A problem space for transportation problem solving

The initial knowledge state

The initial knowledge state for transportation problem solving contains lines which represent rows and columns, cells and opportunities. A cell can be part of a row and a column and a line can be associated with a opportunity.

Operators

1. First-cp: focus on the line =line
   first-cp (=line [=op =op-list])
   (*delete-db (top))
   (*add-db (calc-penalty =line))
   (*add-db (inst =op opportunity))
   (*delete-db consist-of opportunity =op-list))
   (*add-db (consist-of opportunity (*push =op =op-list))))
2. Calc-penalty: calculate a penalty for the line =line
   calc-penalty (=line [=op =cell-1 =par-1 =cell-2 =par-2])
   (*add-db (relate =op =line))
   (*add-db (first =op =cell-1))
   (*add-db (second =op =cell-2))
   (*add-db (value =op (*- =par-1 =par-2))))
3. Next-cp: change focus from =line2 to =line1
   next-cp (=line1 =line2 [=op =op-list])
   (*add-db (calc-penalty =line2))
   (*delete-db (calc-penalty =line1))
   (*add-db (inst =op opportunity))
   (*delete-db (consist-of opportunity =op-list))
   (*add-db (consist-of opportunity (*push =op =op-list))))
4. Select-line: select the line =line-a
   select-line (=line-a [=line-c])
   (*add-db (allocate =line-a))
   (*delete-db (calc-penalty =line-c)))
5. Select-cell: select the cell =cell
   select-cell ([=line] =cell [=op-list])

Figure 6.16 (Continued)

```
        (*delete-opportunity =op-list)
        (*delete-db (allocate =line))
        (*add-db (allocate =line =cell)))
    6. Determine-allocation-amount: determine an amount =number to be
        allocable
      determine-allocation-amount ([=line =cell] =number)
        (*add-db (allocate =line =cell =number))
        (*delete-db (allocate =line =cell)))
    7. Adjust-amount: allocate the determined amount to the selected cell
        and adjust the amounts of related row =line-1 and column =line-2
      adjust-amount (=line-1 [=current-1] =line-2 [=current-2 =line =cell
        =number])
        (*add-db (top))
        (*delete-db (value =cell 0))
        (*add-db (value =cell =number))
        (*delete-db (current-value =line-1 =current-1))
        (*add-db (current-value =line-1 (*- =current-1 =number)))
        (*delete-db (current-value =line-2 =current-2))
        (*add-db (current-value =line-2 (*- =current-2 =number)))
        (*delete-db (allocate =line =cell =number)))
    8. Delete-line: delete the line =line
      delete-line (=line [=cell-list])
        (*delete-db (status =line nil))(*add-db (status =line deleted))
        (*delete-cell =cell-list))
    9. Copy-allocation: take the amount of line =line1 and write the amount
        on the cell =cell
      copy-allocation (=line1 [=current1] =cell [=line2 =current2])
        (*delete-db (current-value =line1 =current1))
        (*add-db (current-value =line1 0))
        (*delete-db (current-value =line2 =current2))
        (*add-db (current-value =line2 (*- =current2 =current1)))
        (*delete-db (value =cell 0))(*add-db (value =cell =current1)))
Goal state : The goal is to reach a state in which all lines are deleted.
```

Figure 6.17. Initial Working Memory and Rules

Working Memory Elements
(consist-of line (c1 c2 c3 r1 r2 r3))
(inst r1 line)(has r1 (e11 e12 e13))(value r1 120)
(current-value r1 120)(status r1 nil)
(inst r2 line)(has r2 (e21 e22 e23))(value r2 80)
(current-value r2 80)(status r2 nil)
(inst r3 line)(has r3 (e31 e32 e33))(value r3 80)
(current-value r3 80)(status r3 nil)
(inst c1 line)(has c1 (e11 e21 e31))(value c1 150)
(current-value c1 150)(status c1 nil)
(inst c2 line)(has c2 (e12 e22 e32))(value c2 70)
(current-value c2 70)(status c2 nil)
(inst c3 line)(has c3 (e13 e23 e33))(value c3 60)
(current-value c3 60)(status c3 nil)
(consist-of cell (e11 e12 e13 e21 e22 e23 e31 e32 e33))
(inst e11 cell)(parameter e11 8)(value e11 0)(status e11 nil)
(partof e11 r1)(partof e11 c1)
(inst e12 cell)(parameter e12 5)(value e12 0)(status e12 nil)
(partof e12 r1)(partof e12 c2)
(inst e13 cell)(parameter e13 6)(value e13 0)(status e13 nil)
(partof e13 r1)(partof e13 c3)
(inst e21 cell)(parameter e21 15)(value e21 0)(status e21 nil)
(partof e21 r2)(partof e21 c1)
(inst e22 cell)(parameter e22 10)(value e22 0)(status e22 nil)
(partof e22 r2)(partof e22 c2)
(inst e23 cell)(parameter e23 12)(value e23 0)(status e23 nil)
(partof e23 r2)(partof e23 c3)
(inst e31 cell)(parameter e31 3)(value e31 0)(status e31 nil)
(partof e31 r3)(partof e31 c1)
(inst e32 cell)(parameter e32 9)(value e32 0)(status e32 nil)
(partof e32 r3)(partof e32 c2)
(inst e33 cell)(parameter e33 10)(value e33 0)(status e33 nil)
(partof e33 r3)(partof e33 c3) (consist-of opportunity nil)

Figure 6.17. (Continued)

```
Rules
    1. (inst =line line)(top)
    (consist-of opportunity =op-list)(bind =op (gensym)); enable action
    ==>
    first-cp (=line =op =op-list)
    2. (inst =line1 line)(calc-penalty =line1)(inst =line2 line)(*no (top))
    (*no (*equal =line1 =line2))
    (relate =some =line1)(*no (relate =some =line2))
    (consist-of opportunity =op-list)(bind =op (gensym))
    ==>
    next-cp (=line1 =line2 =op =op-list)
    3. (inst =line line)(calc-penalty =line)(*no (top))
    (*no (relate =some =line))
    (inst =op opportunity)(*no (relate =op =some)); added heuristic
    (inst =cell-1 cell)(inst =cell-2 cell)(*no (*equal =cell-1 =cell-2))
    (parameter =cell-1 =par-1)(parameter =cell-2 =par-2)
    (partof =cell-1 =line)(partof =cell-2 =line)
    ==>
    calc-penalty (=line =op =cell-1 =par-1 =cell-2 =par-2)
    4. (top)(inst =line1 line)(current-value =line1 =current1)(status =line1 nil)
    (*no (*equal =current 0))(partof =cell =line1)(status =cell nil)(value =cell 0)
    (has =line1 =list)(*only-one status nil =list) ; added heuristic
    (inst =line2 line)(partof =cell =line2)(*no (*equal =line1 =line2))
    (current-value =line2 =current2)
    ==>
    copy-allocation (=line1 =current1 =cell =line2 =current2)
    5. (inst =line1 line)(status =line1 nil)
    (inst =op opportunity)(relate =op =line1)
    (inst =line2 line)(calc-penalty =line2)
    ==>
    select-line (=line-a =line-c)
    6. (inst =line line)(inst =cell cell)(allocate =line)(partof =cell =line)
    (consist-of opportunity =op-list)
```

Figure 6.17. (Continued)

```
==>
select-cell (=line =cell =op-list)
7. (inst =cell cell)(allocate =cell)
(inst =line-1 line)(partof =cell =line-1)(current-value =line-1 =current1)
(inst =line-2 line)(partof =cell =line-2)(current-value =line-2 =current2)
(*no (*equal =line-1 =line-2))
==>
determine-allocation-amount (=line =cell =current-1)
8. (inst =cell cell)(allocate =cell =number)(*numberp =number)
(inst =line-1 line)(partof =cell =line-1)(current-value =line-1 =current1)
(inst =line-2 line)(partof =cell =line-2)(current-value =line-2 =current2)
(*no (*equal =line-1 =line-2))
==>
adjust-amount (=line-1 =current-1 =line-2 =current-2 =line =cell =number)
9. (top)(inst =line line)(current-value =line 0)(status =line nil)
(has =line =cell-list)
==>
delete-line (=line =cell-list)
10. consist-of line =list)(*every status deleted =list)(top)
==>
(*add-db (halt))
```

Figure 6.18. State constraints

A. OPPORTUNITY PRINCIPLE

1. An opportunity value must be calculated for a line which is not deleted.

(opportunity-line-not-deleted.
(inst =line line)(calc-penalty =line)
** (status =line nil))

2. Consider only one line at a time.

(consider-one-line
(inst =line1 line)(calc-penalty =line1)
(inst =line2 line)(calc-penalty =line2)(*no (*equal =line1 =line2))
** nil)

3. Consider only one opportunity at a time.

(consider-one-opportunity
(calc-penalty =line)(inst =op1 opportunity)(inst =op2 opportunity)
(*no (relate =op1 =some))(*no (relate =op2 =some))(*no (*equal =op1
=op2))
** nil)

4. Do not relate more than one opportunity with a line.

(relate-one-opp-with-one-line
(calc-penalty =line)(inst =op1 opportunity)(relate =op1 =line)
(inst =op2 opportunity)(relate =op2 =line)(*no (*equal =op1 =op2))
** nil)

5. Do not relate more than one line with an opportunity.

(relate-one-line-with-one-opp
(calc-penalty =line1)(inst =line2 line)(relate =op =line1)
(relate =op =line2)(*no (*equal =line1 =line2))
** nil)

6. Do not consider a line which is already associated with an opportunity.

(do-not-consider-line-with-op
(calc-penalty =line)(relate =op1 =line)
(inst =op2 opportunity)(*no (relate =op2 =some))(*no (*equal =op1 =op2))
** nil)

Figure 6.18. (Continued)

```
7. A first cell of an opportunity must be part of a line which is associated
with the opportunity.
(first-opportunity-same-line
(inst =line line)(calc-penalty =line)(relate =op =line)
(inst =cell cell)(first =op =cell)
** (partof =cell =line))
8. A second cell of an opportunity must be part of a line which is associated
with the opportunity.
(second-opportunity-same-line
(inst =line line)(calc-penalty =line)(relate =op =line)
(inst =cell cell)(second =op =cell)
** (partof =cell =line))
9. A first cell of an opportunity must be a next-least cost cell in a line
associated with the opportunity.
(opportunity-first-nextleast
(inst =line line)(calc-penalty =line)(relate =op =line)
(inst =cell-1 cell)(first =op =cell-1)(has =line =cell-list)
** (*nextleast parameter =cell-1 (*remove-deleted =cell-list)))
10. A second cell of an opportunity must be a least cost cell in a line
associated with the opportunity.
(opportunity-second-least
(inst =line line)(calc-penalty =line)(relate =op =line)
(inst =cell-2 cell)(second =op =cell-2)(has =line =cell-list)
** (*least parameter =cell-2 (*remove-deleted =cell-list)))
11. A first cell of an opportunity must not be deleted.
(first-opportunity-cell-not-deleted
(inst =op opportunity)(calc-penalty =line)(relate =op =line)
(inst =cell-1 cell)(inst =op opportunity)(first =op =cell-1)
** (status =cell-1 nil))
12. A second cell of an opportunity must not be deleted.
(second-opportunity-cell-not-deleted
(inst =op opportunity)(calc-penalty =line)(relate =op =line)
(inst =cell-2 cell)(inst =op opportunity)(second =op =cell-2)
```

Figure 6.18. (Continued)

** (status =cell-2 nil))

13. Select a line which has the largest opportunity value.

(select-largest-opportunity-line
(inst =line line)(allocate =line)(relate =op =line)
(*no (calc-penalty =some))
(consist-of opportunity =op-list)(*every value =some =op-list)
** (*largest value =op =op-list))

14. Select a line after all lines which are not deleted are associated with some opportunity.

(select-line-after-all-opportunities
(inst =line-1 line)(allocate =line-1)
(inst =line line)(status =line nil)(*no (relate =some =line))
** nil)

B. LEAST COST ALLOCATIONS

1. Select a cell which has the least cost.

(select-least-cost-cell
(inst =cell cell)(inst =line line)(allocate =line =cell)(has =line =cell-list)
** (*least parameter =cell (*remove-deleted =cell-list)))

2. Select a larger current value of two lines of which a selected cell is part.

(select-maximum-amount-for-allocation
(inst =cell cell)(allocate =cell =number)(*numberp =number)
(partof =cell =line-1)(partof =cell =line-2)(*no (*equal =line-1 =line-2))
(current-value =line-1 =val-1)(current-value =line-2 =val-2)(*<= =val-1 =val-2)
** (*equal =number =val-1))

C. DEMAND SUPPLY REQUIREMENT

1. Delete a line whose current value is zero.

(deleted-line-when-all-amount-allocated
(inst =line line)(has =line =cell-list)(status =line deleted)
(value =line =value)(*no (*equal (*total value =cell-list) =value))
** nil)

2. A value of a line must be equal to a current value of the line plus total allocations made to the line.

Figure 6.18. (Continued)

```
(value=total+current
(inst =line line)(has =line =cell-list)(current-value =line =current-value)
(value =line =value)
(*no (*equal =value (*+ (*total value =cell-list) =current-value)))
** nil)
```

D. ORDER PRINCIPLE

1. Select a cell after a line is selected.

```
(select-cell-after-select-line
(inst =cell cell)(allocate =cell)
(inst =line line)(or (calc-penalty =some)(allocate =line)(allocate =some
=number))
** nil)
```

2. Determine an allocation amount for a selected cell after the selection of line and cell.

```
(determine-amount-after-select-line/cell
(inst =cell cell)(allocate =cell =amount)(*numberp =amount)
(or (calc-penalty =some)(allocate =some))
** nil)
```

3. Adjust current values of lines given there is a determined amount.

```
(adjust-amount-after-determine
(inst =line line)(current-value =line 0)(status =line nil)
(or (allocate =some)(allocate =some =amount)(calc-penalty =some))
** nil)
```

4. Delete a line with zero current value after adjustment.

```
(delete-line-after-adjust
(inst =line line)(*no (top))(current-value =line 0)(*no (status =line deleted))
** nil)
```

5. When all cells except one cell are deleted, do not calculate penalties.

```
(prefer-copy-over-calc-penalty
(inst =line line)(current-value =line =current)(status =line nil)
(*no (*equal =current 0))(partof =cell =line)(status =cell nil)
(has =line =list)(*only-one status nil =list)
** (*no (calc-penalty =some)))
```

Figure 6.19. A program trace for the example problem

#S(INSTANCE RULE START ACTIONS NIL VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE FIRST-CP ACTIONS ((*ADD-DB (CONSIST-OF OPPORTUNITY (*PUSH G377 NIL))) (*DELETE-DB (CONSIST-OF OPPORTUNITY NIL)) (*ADD-DB (INST G377 OPPORTUNITY)) (*ADD-DB (CALC-PENALTY R1)) (*DELETE-DB (TOP))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE CALC-PENALTY ACTIONS ((*ADD-DB (VALUE G377 (*- 6 5))) (*ADD-DB (SECOND G377 E12)) (*ADD-DB (FIRST G377 E13)) (*ADD-DB (RELATE G377 R1))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE NEXT-CP ACTIONS ((*ADD-DB (CONSIST-OF OPPORTUNITY (*PUSH G396 (G377)))) (*DELETE-DB (CONSIST-OF OPPORTUNITY (G377))) (*ADD-DB (INST G396 OPPORTUNITY)) (*ADD-DB (CALC-PENALTY C1)) (*DELETE-DB (CALC-PENALTY R1))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE CALC-PENALTY ACTIONS ((*ADD-DB (VALUE G392 (*- 9 3))) (*ADD-DB (SECOND G392 E31)) (*ADD-DB (FIRST G392 E32)) (*ADD-DB (RELATE G392 R3))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE NEXT-CP ACTIONS ((*ADD-DB (CONSIST-OF OPPORTUNITY (*PUSH G262 (G392 G377)))) (*DELETE-DB (CONSIST-OF OPPORTUNITY (G392 G377))) (*ADD-DB (INST G262 OPPORTUNITY)) (*ADD-DB (CALC-PENALTY C1)) (*DELETE-DB (CALC-PENALTY R3))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE CALC-PENALTY ACTIONS ((*ADD-DB (VALUE G262 (*- 8 3))) (*ADD-DB (SECOND G262 E31)) (*ADD-DB (FIRST G262 E11)) (*ADD-DB (RELATE G262 C1))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE NEXT-CP ACTIONS ((*ADD-DB (CONSIST-OF OPPORTUNITY (*PUSH G242 (G262 G392 G377)))) (*DELETE-DB (CONSIST-OF OPPORTUNITY (G262 G392 G377))) (*ADD-DB (INST G242 OPPORTUNITY)) (*ADD-DB (CALC-PENALTY C2)) (*DELETE-DB (CALC-PENALTY C1))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE CALC-PENALTY ACTIONS ((*ADD-DB (VALUE G242 (*- 9 5))) (*ADD-DB (SECOND G242 E12)) (*ADD-DB (FIRST G242 E32)) (*ADD-DB (RELATE G242 C2))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE NEXT-CP ACTIONS ((*ADD-DB (CONSIST-OF OPPORTUNITY (*PUSH G253 (G242 G262 G392 G377)))) (*DELETE-DB (CONSIST-OF OPPORTUNITY (G242 G262 G392 G377))) (*ADD-DB (INST G253 OPPORTUNITY)) (*ADD-DB (CALC-PENALTY C3)) (*DELETE-DB (CALC-PENALTY C2))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE CALC-PENALTY ACTIONS ((*ADD-DB (VALUE G253 (*- 10 6))) (*ADD-DB (SECOND G253 E13)) (*ADD-DB (FIRST G253 E33)) (*ADD-DB (RELATE G253 C3))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE SELECT-LINE ACTIONS ((*ADD-DB (ALLOCATE R3)) (*DELETE-DB (CALC-PENALTY C3))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE SELECT-CELL ACTIONS ((*DELETE-DB (ALLOCATE R3)) (*ADD-DB (ALLOCATE R3 E31)) (*DELETE-OPPORTUNITY (G253 G242 G262 G392 G377 G366))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE DETERMINE-ALLOCATION-AMOUNT ACTIONS ((*DELETE-DB (ALLOCATE E31)) (*ADD-DB (ALLOCATE E31 80))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE ADJUST-AMOUNT ACTIONS ((*DELETE-DB (ALLOCATE E31 80)) (*ADD-DB (CURRENT-VALUE C1 (*- 150 80))) (*DELETE-DB (CURRENT-VALUE C1 150)) (*ADD-DB (CURRENT VALUE R3 (*- 80 80))) (*DELETE-DB (CURRENT-VALUE R3 80)) (*ADD-DB (VALUE E31 80)) (*DELETE-DB (VALUE E31 0)) (*ADD-DB (TOP))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE DELETE-LINE ACTIONS ((*DELETE-CELL (E31 E32 E33)) (*ADD-DB (STATUS R3 DELETED)) (*DELETE-DB (STATUS R3 NIL))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE FIRST-CP ACTIONS ((*ADD-DB (CONSIST-OF OPPORTUNITY (*PUSH G275 NIL))) (*DELETE-DB (CONSIST-OF OPPORTUNITY NIL)) (*ADD-DB (INST G275 OPPORTUNITY)) (*ADD-DB (CALC-PENALTY R2)) (*DELETE-DB (TOP))) VIOLATIONS NIL TOTAL 0)

Figure 6.19. (Continued)

#S(INSTANCE RULE CALC-PENALTY ACTIONS ((*ADD-DB (VALUE G275 (*- 12 10))) (*ADD-DB (SECOND G275 E22)) (*ADD-DB (FIRST G275 E23)) (*ADD-DB (RELATE G275 R2))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE NEXT-CP ACTIONS ((*ADD-DB (CONSIST-OF OPPORTUNITY (*PUSH G286 (G275)))) (*DELETE-DB (CONSIST-OF OPPORTUNITY (G275))) (*ADD-DB (INST G286 OPPORTUNITY)) (*ADD-DB (CALC-PENALTY R1)) (*DELETE-DB (CALC-PENALTY R2))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE CALC-PENALTY ACTIONS ((*ADD-DB (VALUE G286 (*- 6 5))) (*ADD-DB (SECOND G286 E12)) (*ADD-DB (FIRST G286 E13)) (*ADD-DB (RELATE G286 R1))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE NEXT-CP ACTIONS ((*ADD-DB (CONSIST-OF OPPORTUNITY (*PUSH G297 (G286 G275)))) (*DELETE-DB (CONSIST-OF OPPORTUNITY (G286 G275))) (*ADD-DB (INST G297 OPPORTUNITY)) (*ADD-DB (CALC-PENALTY C3)) (*DELETE-DB (CALC-PENALTY R1))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE CALC-PENALTY ACTIONS ((*ADD-DB (VALUE G297 (*- 12 6))) (*ADD-DB (SECOND G297 E13)) (*ADD-DB (FIRST G297 E23)) (*ADD-DB (RELATE G297 C3))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE NEXT-CP ACTIONS ((*ADD-DB (CONSIST-OF OPPORTUNITY (*PUSH G308 (G297 G286 G275)))) (*DELETE-DB (CONSIST-OF OPPORTUNITY (G297 G286 G275))) (*ADD-DB (INST G308 OPPORTUNITY)) (*ADD-DB (CALC-PENALTY C2)) (*DELETE-DB (CALC-PENALTY C3))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE CALC-PENALTY ACTIONS ((*ADD-DB (VALUE G308 (*- 10 5))) (*ADD-DB (SECOND G308 E12)) (*ADD-DB (FIRST G308 E22)) (*ADD-DB (RELATE G308 C2))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE NEXT-CP ACTIONS ((*ADD-DB (CONSIST-OF OPPORTUNITY (*PUSH G326 (G308 G297 G286 G275)))) (*DELETE-DB (CONSIST-OF OPPORTUNITY (G308 G297 G286 G275))) (*ADD-DB (INST G326 OPPORTUNITY)) (*ADD-DB (CALC-PENALTY C1)) (*DELETE-DB (CALC-PENALTY C2))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE CALC-PENALTY ACTIONS ((*ADD-DB (VALUE G326 (*- 15 8))) (*ADD-DB (SECOND G326 E11)) (*ADD-DB (FIRST G326 E21)) (*ADD-DB (RELATE G326 C1))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE SELECT-LINE ACTIONS ((*ADD-DB (ALLOCATE C1)) (*DELETE-DB (CALC-PENALTY C1))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE SELECT-CELL ACTIONS ((*DELETE-DB (ALLOCATE C1)) (*ADD-DB (ALLOCATE E11)) (*DELETE-OPPORTUNITY (G326 G308 G297 G286 G275))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE DETERMINE-ALLOCATION-AMOUNT ACTIONS ((*DELETE-DB (ALLOCATE E11)) (*ADD-DB (ALLOCATE E11 70))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE ADJUST-AMOUNT ACTIONS ((*DELETE-DB (ALLOCATE E11 70)) (*ADD-DB (CURRENT-VALUE R1 (*- 120 70))) (*DELETE-DB (CURRENT-VALUE R1 120)) (*ADD-DB (CURRENT-VALUE C1 (*- 70 70))) (*DELETE-DB (CURRENT-VALUE C1 70)) (*ADD-DB (VALUE E11 70)) (*DELETE-DB (VALUE E11 0)) (*ADD-DB (TOP))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE DELETE-LINE ACTIONS ((*DELETE-CELL (E11 E21 E31)) (*ADD-DB (STATUS C1 DELETED)) (*DELETE-DB (STATUS C1 NIL))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE FIRST-CP ACTIONS ((*ADD-DB (CONSIST-OF OPPORTUNITY (*PUSH G354 NIL))) (*DELETE-DB (CONSIST-OF OPPORTUNITY NIL)) (*ADD-DB (INST G354 OPPORTUNITY)) (*ADD-DB (CALC-PENALTY C2)) (*DELETE-DB (TOP))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE CALC-PENALTY ACTIONS ((*ADD-DB (VALUE G354 (*- 10 5))) (*ADD-DB (SECOND G354 E12)) (*ADD-DB (FIRST G354 E22)) (*ADD-DB (RELATE G354 C2))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE NEXT-CP ACTIONS ((*ADD-DB (CONSIST-OF OPPORTUNITY (*PUSH G359 (G354)))) (*DELETE-DB (CONSIST-OF OPPORTUNITY (G354))) (*ADD-DB (INST G359 OPPORTUNITY)) (*ADD-DB (CALC-PENALTY C3)) (*DELETE-DB (CALC-PENALTY C2))) VIOLATIONS NIL TOTAL 0)

Figure 6.19. (Continued)

#S(INSTANCE RULE CALC-PENALTY ACTIONS ((*ADD-DB (VALUE G359 (*- 12 6))) (*ADD-DB (SECOND G359 E13)) (*ADD-DB (FIRST G359 E23)) (*ADD-DB (RELATE G359 C3))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE NEXT-CP ACTIONS ((*ADD-DB (CONSIST-OF OPPORTUNITY (*PUSH G364 (G359 G354)))) (*DELETE-DB (CONSIST-OF OPPORTUNITY (G359 G354))) (*ADD-DB (INST G364 OPPORTUNITY)) (*ADD-DB (CALC-PENALTY R1)) (*DELETE-DB (CALC-PENALTY C3))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE CALC-PENALTY ACTIONS ((*ADD-DB (VALUE G364 (*- 6 5))) (*ADD-DB (SECOND G364 E12)) (*ADD-DB (FIRST G364 E13)) (*ADD-DB (RELATE G364 R1))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE NEXT-CP ACTIONS ((*ADD-DB (CONSIST-OF OPPORTUNITY (*PUSH G369 (G364 G359 G354)))) (*DELETE-DB (CONSIST-OF OPPORTUNITY (G364 G359 G354))) (*ADD-DB (INST G369 OPPORTUNITY)) (*ADD-DB (CALC-PENALTY R2)) (*DELETE-DB (CALC-PENALTY R1))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE CALC-PENALTY ACTIONS ((*ADD-DB (VALUE G369 (*- 12 10))) (*ADD-DB (SECOND G369 E22)) (*ADD-DB (FIRST G369 E23)) (*ADD-DB (RELATE G369 R2))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE SELECT-LINE ACTIONS ((*ADD-DB (ALLOCATE C3)) (*DELETE-DB (CALC-PENALTY R2))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE SELECT-CELL ACTIONS ((*DELETE-DB (ALLOCATE C3)) (*ADD-DB (ALLOCATE E13)) (*DELETE-OPPORTUNITY (G369 G364 G359 G354))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE DETERMINE-ALLOCATION-AMOUNT ACTIONS ((*DELETE-DB (ALLOCATE E13)) (*ADD-DB (ALLOCATE E13 50))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE ADJUST-AMOUNT ACTIONS ((*DELETE-DB (ALLOCATE E13 50)) (*ADD-DB (CURRENT-VALUE R1 (*- 50 50))) (*DELETE-DB (CURRENT-VALUE R1 50)) (*ADD-DB (CURRENT-VALUE C3 (*- 60 50))) (*DELETE-DB (CURRENT-VALUE C3 60)) (*ADD-DB (VALUE E13 50)) (*DELETE-DB (VALUE E13 0)) (*ADD-DB (TOP))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE DELETE-LINE ACTIONS ((*DELETE-CELL (E11 E12 E13)) (*ADD-DB (STATUS R1 DELETED)) (*DELETE-DB (STATUS R1 NIL))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE COPY-ALLOCATION ACTIONS ((*ADD-DB (VALUE E23 10)) (*DELETE-DB (VALUE E23 0)) (*ADD-DB (CURRENT-VALUE R2 (*- 80 10))) (*DELETE-DB (CURRENT-VALUE R2 80)) (*ADD-DB (CURRENT-VALUE C3 0)) (*DELETE-DB (CURRENT-VALUE C3 10))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE COPY-ALLOCATION ACTIONS ((*ADD-DB (VALUE E22 70)) (*DELETE-DB (VALUE E22 0)) (*ADD-DB (CURRENT-VALUE R2 (*- 70 70))) (*DELETE-DB (CURRENT-VALUE R2 70)) (*ADD-DB (CURRENT-VALUE C2 0)) (*DELETE-DB (CURRENT-VALUE C2 70))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE DELETE-LINE ACTIONS ((*DELETE-CELL (E21 E22 E23)) (*ADD-DB (STATUS R2 DELETED)) (*DELETE-DB (STATUS R2 NIL))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE DELETE-LINE ACTIONS ((*DELETE-CELL (E13 E23 E33)) (*ADD-DB (STATUS C3 DELETED)) (*DELETE-DB (STATUS C3 NIL))) VIOLATIONS NIL TOTAL 0)
#S(INSTANCE RULE DELETE-LINE ACTIONS ((*DELETE-CELL (E12 E22 E32)) (*ADD-DB (STATUS C2 DELETED)) (*DELETE-DB (STATUS C2 NIL))) VIOLATIONS NIL TOTAL 0)

6.6 Summary

In this chapter, PSCD is formally presented. PSCD is based on problem space representation, and consists of three components: productions, state constraints, and a strengthening process. PSCD is applied to a counting task as an initial validation of its performance ability, and to a transportation problem which is a main task of this study. PSCD is shown to be sufficient to perform a transportation problem solving as well as counting.

Chapter 7 VALIDATION OF THE MODEL

7.1 Validation Methodology

The validation methodology employed in this study is the criterion of *sufficiency* (Newell, 1973a; Ohlsson and Rees, 1991). Newell (1973a) observed two constructions that had dominated in traditional psychology experimental studies: (a) at a low level, the discovery and empirical exploration of phenomena and (b) at the middle level, the formulation of questions to be put to nature that center on the resolution of binary oppositions. Newell argued that experimental studies at the low and middle level typically tried to formulate a local theory from observing phenomena and attempted with local methods to falsify the theory or provide evidence for the failure of falsification. What is missing, it is argued, is a frame that constrains what other methods might also be evoked to perform the same task. In short, they did not consider the total method which would model the control structure as well. Newell argued for the development of a grand theory which might be driven by general concerns such as exploring learning or development.

Newell (1973a) suggested three strategies for developing a high level of grand theory. The first was to build *complete processing models* for individual tasks. Simulations must specify the detailed control structure and detailed assumptions about memories and elementary processes, and must be sufficient to reproduce data. This sufficiency criterion reduces the degrees of freedom that allow so many models to coexist with the same data. Newell's second suggestion was to *analyze a complex task* and do all of it. The aim is to demonstrate that one has a sufficient theory of human behavior. Again the aim was to reduce the range of possible models consistent with the data, in this case by increasing the amount and variety of data for which the models must

account. The third suggested strategy is to build a complete processing model that perform *many complex tasks*.

In his recent book, Newell (1990) elaborates on his position about the grand theory approach and further argues that building a large scale program which would put together and synthesize our existing understanding of cognition will provide a new way for psychology to leap forward. Newell explains his position for *unified theories of cognition* as follows:

> The task at hand is to try to get *some* candidate theories that have a large enough scope in order to get the gains inherent in such unification, and to show us all that they are real. The task is somehow to cajole ourselves into putting it all together, even though we don't know many of the mechanisms that are operating. The long run can be entrusted to take care of the eventual emergence of a single unified theory and how that convergence or selection will take place. (Newell, 1990, p. 17)

Newell continues to argue that:

> If there is any property that a theory of cognition must explain it is how intelligence is actually possible. Necessary characteristics are well and good, but they are substantially less than half the story. *Sufficiency* is all-important. Intelligence itself is a sufficiency of capability. To be intelligent is to be able to do certain things.... (Newell, 1990, p. 158)

From these remarks, it can be said that the objective of building a large scale program is to show that it is indeed capable of doing certain things that are wide in scope. Newell's arguments for the grand theory and the sufficiency criterion actually have a root in his belief of what a theory is.

According to Newell (1990), a theory is some body of explicit knowledge, from which answers can be obtained to questions by inquiries. Some answers might be predictions, some might be explanations, some might be prescriptions

for control. If this body of knowledge yields answers to those questions, it can be called a theory. The essential role of a theory, be it just a collection of facts or the fundamental axioms of a subject matter, is to provide needed answers. Newell then defined a theory as an explicit body of knowledge, from which answers can be obtained by anyone skilled in the art. From this perspective, Newell explained properties associated with a theory. Theories are not objects of discrimination, but of *approximation*. Theories cannot be absolute because the world can't be known with absolute certainty. Rather theories approximate the truth in nature. Usefulness often traded off against truth. Theories that are known to be wrong continue to be used, because they are the best available, for instance Fitts' law. Theories are treated as a tool to be used for some externally useful purpose. Theories *cumulate*. They are refined and reformulated, corrected and expanded. Theories are not just falsified but refined and expanded.

Newell's position about a theory and a methodology can be summarized as follows. The role of a theory lies in its usefulness as well as its truth. We must build a large-scale program which integrates as many known things as possible, and demonstrate that it is sufficient to perform diverse and complex tasks. Doing this way yields several benefits: bearing more constraints; increase in identifiability; avoiding irrelevant-specification problem; and allowing practical application (Newell, 1990, pp. 21-23). His formulation can solve such methodological problems as identifiability and irrelevant-specification problems and also opens a way for application. His formulation merits attention, at least from cognitive scientists who have applications in mind, not only because of its methodological advantages but also because of its possibilities for application.

Although Newell's formulation has received wide acceptance (Anderson, 1983; VanLehn, 1991), Simon (1989) pointed out that we still needed the collection and analysis of empirical data to help us decide alternative models of performance and to eventually find a small set of basic mechanisms underlying more comprehensive models. Simon contrasted alternative research styles: Newell's style which prefers top-down theory construction from a hypothesis about the control system to build a highly homogeneous cognitive mechanism, and the position of traditional experimental psychologists who prefers bottom-up construction from careful quantitative observation of phenomena to build a specialized component for performing a particular range of tasks. Although Simon agreed on the difficulties of resolving conflicting empirical data soon and arriving at a unified picture of the HIP system just with specialized components, he argued that we must recognize usefulness of empirical data in generating quantitative invariants on which such a unified system must be based. What Simon says is that the necessary criterion also counts.

Simon's observation on contrasting methodologies has similar repercussions of Miller's account (Miller, 1978). Miller differentiated two competing methodologies, theory demonstration and theory development. Theory demonstration is concerned with validating a theory through empirical data, whereas theory development involves building a theory into a computer model and showing that the theory is sufficient to account for the phenomenon under study. In addition to these two competing methodologies, there can be some other types of methodologies that have been used in AI studies.

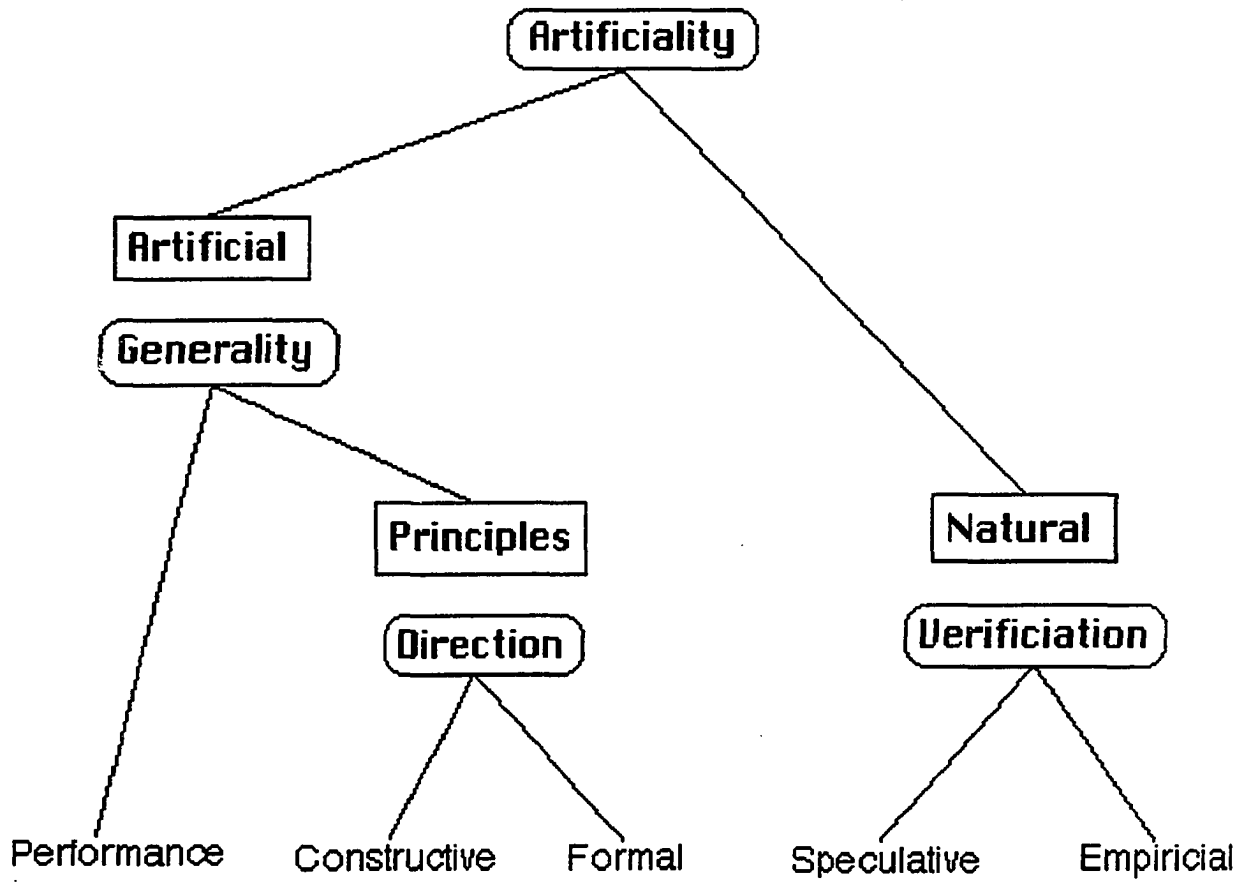
Hall and Kibler (1985) described and organized AI methodologies into five categories (Figure 7.1). They used two dimensions for classifying AI methodologies. The first of two dimensions is artificiality of the computational

mechanisms under study. The issue of the artificiality differentiates AI studies which are undertaken with the intent of exploring human cognitive phenomena from studies of intelligent functioning by any computational mechanisms possible. The second that can differentiate differing methodologies is the generality of reasoning methods sought. This dimension of generality distinguishes between studies of developing computation techniques for a specific applications and research aimed at discovering general reasoning mechanisms. They recognized that the classification by these two dimensions could be incompletely filled because they could not find a study that could fit the cell formed by specific reasoning mechanisms and human cognitive phenomena, and also admitted that the some of AI studies would not be unambiguously categorized under the proposed classification scheme because researchers usually shifted between and shared differing perspectives. Even with these seemingly ambiguity and inexactness, Hall and Kibler's classification appeals to be of great value in understanding differing methodologies and also in positioning our study in an appropriate perspective from the total picture of AI methodologies.

Viewing from the perspective of Hall and Kibler's categorization, this study can be put under the "natural" category because this study follows the traditions of cognitive modeling. Under the label of natural, there are two subcategories: empirical and speculative. The empirical study typically demonstrates correspondence between behavior of artificial and natural systems, whereas the speculative study presents an analysis of natural intelligence but does not make rigorous efforts to demonstrate empirical evidence for correspondence between artificial and natural systems. This study is empirical to the extent that PSCD is demonstrated to show its abilities to

perform and to make diagnosis in the same way as shown in empirical data. This study is also speculative to the extent that it does not validate but simply accepts the hypothesis that symbolic search space (Newell and Simon, 1972; Newell, 1990) is the right paradigm in the domain of management science as

Figure 7.1. Hall and Kibler's (1985, p. 171) typology



well, and that bugs that can be found in transportation problem solving are mainly originated from students' attempts to derive procedures from inadequate conceptual knowledge as in bugs found in the domain of counting and arithmetics (Resnick, 1982, 1983; Greeno et al., 1984; Ohlsson and Rees, 1991). This study is empirical in some parts and speculative in other aspects. Although Hall and Kibler's classification scheme is not able to position our study unambiguously in one perspective, their scheme is still of value because this study's methodological perspectives can be *explicitly* stated with their scheme.

With Simon's (1989) distinction between top-down and bottom-up constructions, these two perspectives may be added as subcategories under the empirical category. This study approaches the problem in the top-down fashion. This study initially delineates the functional architecture, PSCD, for cognitive diagnosis with heavy reliance on previous works (Langley, 1985; Langley et al., 1990; Anderson, 1983, 1989; Newell, 1990; Gelman and Meck, 1986; Ohlsson and Rees, 1991; Greeno et al., 1984). Most cognitive theories are built into PSCD. Then PSCD is applied to transportation problem solving tasks to demonstrate its capabilities of performance and diagnosis.

7.2 Data Collection

Data are problem-solving traces of students. Collected data are used to be compared with PSCD's problem solving traces and also to test PSCD for its ability to make correct diagnosis given students' incorrect traces. A test with three transportation problems (see Figure 7.2) was made and given to students as a class quiz. All problems required students to derive initial solutions by the VAM method. Before giving the quiz, instructors explained the concepts

underlying the VAM method and gave an example of it. Right after this instruction, the quiz was distributed to students. Students were asked to show

Figure 7.2. Transportation problems

Problem 1. Set up the initial table by VAM method

From \ To	DEMAND 1	DEMAND 2	DEMAND 3	Supply
SUPPLY 1	8	5	6	120
SUPPLY 2	15	10	12	80
SUPPLY 3	3	9	10	80
Demand	150	70	60	280

Figure 7.2. (Continued)

Problem 2. Set up the initial table by VAM method

	DEMAND 1	DEMAND 2	DEMAND 3	DEMAND 4	
SUPPLY 1	10	0	20	11	20
SUPPLY 2	12	7	9	20	10
SUPPLY 3	0	14	16	18	5
	5	15	15	10	

Problem 3. Set up the initial table by VAM method

	DEMAND 1	DEMAND 2	DEMAND 3	DEMAND 4	
SUPPLY 1	12	10	9	15	36
SUPPLY 2	10	8	2	10	25
SUPPLY 3	9	5	15	8	30
SUPPLY 4	0	0	0	0	30
	26	40	25	30	

each solution steps explicitly. In order to provide enough space for them to write every steps additional blank transportation tableau were provided to each problem. Students were urged to finish the quiz and to use any procedure that would make sense to them even if their understanding of the VAM method was minimal. Students were informed that the quiz counted on their grades and asked to do their best.

The quiz was administered in three classes of MGT 245. Since MGT 245 is the introductory course of management science, it is assumed that students do not have general background knowledge of management science that can be transferred and used in transportation problem solving. In other words, it is assumed that student errors arise only from incomplete understanding of principles of transportation problem solving, not from buggy procedures that result from attempts of using similar knowledge of management science (e.g., Linear Programming). Data from the first class were largely used for tuning PSCD's working memory elements, rules, and constraints that were initially developed from the description of the VAM method in the textbook (Lee et al., 1990). The number of data items (i.e., number of answer sheets that show incorrect solutions) collected from the other two classes was 38. The next section presents the results of the analysis and explanations.

7.3 Results and Explanations

Reviewing the data revealed error patterns that can be usefully grouped in terms of violations of state constraints. In order to keep the discussion focused, this study will use the first problem in the quiz to illustrate five cases. The first step was to translate problem solving traces shown in answer sheets into action protocol. An action is a list which has the name of action as the first

element and a set of arguments that follows the name. Arguments of each operator show the specific instantiation (i.e., variable bindings) of the operator. For example, in Figure 7.3 the first action in the action protocol is (first-cp r1 op1 nil). The action shows that a student takes the r1 as the first line to calculate a penalty.

Figure 7.3 shows the partial answer in the form of transportation tableau, the action protocol of subject-1, and diagnosis results of PSCD. Each iteration of VAM procedure is separated by a dotted line to make the protocol more readable. The action protocol is the input to PSCD and PSCD generates a report showing violated constraints, if any. The report generated by PSCD is edited and shown in the context of the original action protocol. Lists which have constraint names in capital letters show violated constraints of the respective state. Subject-1 appears to have difficulties in in applying two constraints, FIRST-OPPORTUNITY-CELL-NOT-DELETED and OPPORTUNITY-FIRST-NEXTLEAST, during his problem solving process. Apparently these two constraints were not considered when they should have been. His error is calculating penalties with deleted cells. Curiously he did not violate these constraints at the first iteration but committed violations at the consecutive two iterations. Given this apparent inconsistency, it may be argued that this error is not related to principled understanding but can be viewed as just a slip. Some researchers argued for *slips* as the main cause of most errors (Anderson and Jeffries, 1985), and others distinguished *mistakes* from slips and attempted to model the process that might cause mistakes (Brown and VanLehn, 1980). Yet it is also argued that the distinction between slips and mistakes may not be as clear as expected (Payne and Squibb, 1990).

Figure 7.3. The action protocol of subject-1

From \ To	DEMAND 1	DEMAND 2	DEMAND 3	Supply
SUPPLY 1	8 (70)	5	6	120
SUPPLY 2	15	10	12	80
SUPPLY 3	3 80	9 ———	10 ———	80
Demand	150	70	60	280

(7) (15 - 8) 4 (9 - 5) 4 (10 - 6)

```
(first-cp 'r1 'op1 nil)
(calc-penalty 'r1 'op1 'e13 6 'e12 5)
(next-cp 'r1 'r2 'op2 '(op1))
(calc-penalty 'r2 'op2 'e23 12 'e22 10)
(next-cp 'r2 'r3 'op3 '(op2 op1))
(calc-penalty 'r3 'op3 'e32 9 'e31 3)
(next-cp 'r3 'c1 'op4 '(op3 op2 op1))
(calc-penalty 'c1 'op4 'e11 8 'e31 3)
(next-cp 'c1 'c2 'op5 '(op4 op3 op2 op1))
(calc-penalty 'c2 'op5 'e32 9 'e12 5)
(next-cp 'c2 'c3 'op6 '(op5 op4 op3 op2 op1))
(calc-penalty 'c3 'op6 'e33 10 'e13 6)
(select-line 'r3 'c3)
(select-cell 'r3 'e31 '(op6 op5 op4 op3 op2 op1))
(determine-allocation-amount 'r3 'e31 80)
(adjust-amount 'r3 80 'c1 150 'r3 'e31 80)
(delete-line 'r3 '(e31 e32 e33))
;-----------------------------------------
(first-cp 'r1 'op1 nil)
(calc-penalty 'r1 'op1 'e13 6 'e12 5)
(next-cp 'r1 'r2 'op2 '(op1))
(calc-penalty 'r2 'op2 'e23 12 'e22 10)
(next-cp 'r2 'c1 'op3 '(op2 op1))
(calc-penalty 'c1 'op3 'e21 15 'e11 8)
(next-cp 'c1 'c2 'op4 '(op3 op2 op1))
(calc-penalty 'c2 'op4 'e32 9 'e12 5)
; (FIRST-OPPORTUNITY-CELL-NOT-DELETED OPPORTUNITY-FIRST-NEXTLEAST)
(next-cp 'c2 'c3 'op5 '(op4 op3 op2 op1))
(calc-penalty 'c3 'op5 'e33 10 'e13 6)
;(FIRST-OPPORTUNITY-CELL-NOT-DELETED OPPORTUNITY-FIRST-NEXTLEAST)
(select-line 'c1 'c3)
```

Figure 7.3. (Continued)

```
(select-cell 'c1 'e11 '(op5 op4 op3 op2 op1))
(determine-allocation-amount 'c1 'e11 70)
(adjust-amount 'c1 70 'r1 120 'c1 'e11 70)
(delete-line 'c1 '(e11 e21 e31))
;-----------------------------------------
(first-cp 'r1 'op1 nil)
(calc-penalty 'r1 'op1 'e13 6 'e12 5)
(next-cp 'r1 'r2 'op2 '(op1))
(calc-penalty 'r2 'op2 'e23 12 'e22 10)
(next-cp 'r2 'c2 'op3 '(op2 op1))
(calc-penalty 'c2 'op3 'e32 9 'e12 5)
;(FIRST-OPPORTUNITY-CELL-NOT-DELETED OPPORTUNITY-FIRST-NEXTLEAST)
(next-cp 'c2 'c3 'op4 '(op3 op2 op1))
(calc-penalty 'c3 'op4 'e33 10 'e13 6)
;(FIRST-OPPORTUNITY-CELL-NOT-DELETED OPPORTUNITY-FIRST-NEXTLEAST)
(select-line 'c3 'c3)
(select-cell 'c3 'e13 '(op4 op3 op2 op1))
(determine-allocation-amount 'c3 'e13 50)
(adjust-amount 'r1 50 'c3 60 'c3 'e13 50)
(delete-line 'r1 '(e11 e12 e13))
;-----------------------------------------
(copy-allocation 'c2 70 'e22 'r2 80)
(copy-allocation 'c3 10 'e23 'r2 10)
```

The distinction between slips and mistakes is currently widely accepted in the literature on cognitive errors (Anderson and Jeffries, 1985; Norman, 1981; Kowalski and VanLehn, 1988; Langley et al., 1990; Reason, 1990). The distinguishing factor between these two types of errors is intention. Slips occur when actions deviate from current intention due to execution failures and/or storage failures. In mistakes, actions may run according to plan, but where the plan is inadequate to achieve its desired outcomes. Thus, slips are unintended actions, whereas mistakes are intended but mistaken actions. Both are the same in that they fail to achieve their original goals.

It has been questioned whether errors that are found are mostly slips or mistakes. Anderson and Jeffries (1985) reported that, in the study of novice LISP programmers, the majority of errors were slips due to losses of information in working memory rather than mistakes. Brown and Burton (1978), however, showed that most arithmetic errors made by children were systematic and could be explained by bugs. Bugs are erroneous variants of a procedure and thus denote intentional actions (see 5.1.1 for additional discussion about bugs). Bugs are constructed through purely syntactic manipulation of symbols (Young and O'Shea, 1981; Brown and VanLehn, 1980). Given the massive amount of data and conceptual works that support the constructivist assumption and the importance of conceptual understanding in understanding performance (see 4.3), a need arose to consider a process of semantic rationalization in interpreting students' errors. Errors that result from faulty semantic rationalization process are differentiated from bugs and are called *misconception* (Langley et al., 1990). A misconception involves a person's beliefs about the world and implies faulty understanding, whereas a bug is inherently procedural and implies faulty performance. Langley et al. (1990)

differentiated slips from errors, and further refined errors between bugs and misconception. Since it is believed that misconceptions are fundamental errors that cause bugs to happen, the study focuses on diagnosing misconceptions.

An interesting question becomes whether a slip is easily distinguishable from an error. Examining algebra mal-rules, Payne and Squibb (1990) argued that mal-rules might not be characterized as the occurrence of a mistake but the co-occurrence of a slip and a mistake. In other words, there is "pure" mistake in their errors. They pointed out two implications. First, the slip/mistake distinction is not like all-or-nothing, either students know the right rule but slip in its execution, or they do not know the right rule and must perform local problem solving, or apply an incorrect version of the rule. Rather, errors arise when students are aware of the right rule in some degree but not strong enough to use it. In this case, an appropriate computational mechanism is a strengthening mechanism (Langley, 1985). There can be a case that a weaker faulty rule is preferred to a strong correct rule. For this case, a partial matching system can be used in addition to a strengthening mechanism (Anderson, 1983; Reason, 1990).

The second implication is that new (mal-)rules may arise when a student attempts to make sense of his current procedure. Here again the role of semantics is emphasized in explaining error generation. Payne and Squibb (1990) then proposed a theoretical framework for explaining algebra errors. The main insights are twofolds. First, errors can be modelled by a strengthening mechanism that adjusts the applicability of rules. Second, the value of strength is determined by rule usages and also by semantic rationalization of rules. PSCD is based on these notions. AS explained earlier, PSCD assumes the crudest strengthening mechanism which hypothesizes only existence or non-

existence of constraints in a student model. The value of strength associated with a constraint is determined by its satisfaction or non-satisfaction. The frequency of violations can be also calculated, if needed by the tutoring model.

From this diversion, it may become clear that it does not matter if subject-1's errors are slips or mistakes, but what matters is that his errors can be usefully viewed as violations of two principles. In addition, the cause of violations can be explained as lack of strength associated with constraints.

The rest of the data in Figure 7.4 to Figure 7.7 also shows students' errors as violations of some principles. Subject-2 in Figure 7.4 calculated a penalty for a deleted line, subject-3 in Figure 7.5 did not take the least-cost cell into consideration in calculating a penalty, subject-4 failed to calculate penalties for all undeleted lines, and subject-5 made many different types of errors. Examining original answer sheets that were corrected by instructors, it was found that PSCD's diagnosis was generally compatible with instructors' corrections.

7.4 Summary

In Chapter 6, PSCD is shown to be sufficient to perform transportation problem solving as well as counting. In this chapter, PSCD is applied to the task of diagnosis. Input data to PSCD are action protocols. An action protocol is a sequence of actions with appropriate arguments. Output of PSCD is violated constraints.

To examine the PSCD's diagnosis capability, students' answer sheets to VAM questions were collected in three introductory management science classes. Students' answers were transcribed to action protocols. PSCD was run in the diagnosis mode with action protols and reported constraint violations

are gathered. PSCD's diagnoses and instructors' corrections were compared. It was found that PSCD's diagnoses were generally compatible with instructors' corrections. It appears that the set of constraints hypothesized for VAM problem solving is sufficient to make diagnosis of most errors that can happen in VAM problem solving.

Figure 7.4. The action protocol of subject-2

From \ To	DEMAND 1	DEMAND 2	DEMAND 3	Supply	
SUPPLY 1	8 (70)	5	6	120	1
SUPPLY 2	15	10	12	80	2
SUPPLY 3	3 80	9 ——	10 ——	80	6
Demand	150	70	60	280	
	(7)	5	6		

```
(first-cp 'r1 'op1 nil)
(calc-penalty 'r1 'op1 'e13 6 'e12 5)
(next-cp 'r1 'r2 'op2 '(op1))
(calc-penalty 'r2 'op2 'e23 12 'e22 10)
(next-cp 'r2 'r3 'op3 '(op2 op1))
(calc-penalty 'r3 'op3 'e32 9 'e31 3)
(next-cp 'r3 'c1 'op4 '(op3 op2 op1))
(calc-penalty 'c1 'op4 'e11 8 'e31 3)
(next-cp 'c1 'c2 'op5 '(op4 op3 op2 op1))
(calc-penalty 'c2 'op5 'e32 9 'e12 5)
(next-cp 'c2 'c3 'op6 '(op5 op4 op3 op2 op1))
(calc-penalty 'c3 'op6 'e33 10 'e13 6)
(select-line 'r3 'c3)
(select-cell 'r3 'e31 '(op6 op5 op4 op3 op2 op1))
(determine-allocation-amount 'r3 'e31 80)
(adjust-amount 'r3 80 'c1 150 'r3 'e31 80)
(delete-line 'r3 '(e31 e32 e33))
;-----------------------------------------
(first-cp 'r1 'op1 nil)
(calc-penalty 'r1 'op1 'e13 6 'e12 5)
(next-cp 'r1 'r2 'op2 '(op1))
(calc-penalty 'r2 'op2 'e23 12 'e22 10)
(next-cp 'r2 'r3 'op3 '(op2 op1)); error - cal-penalty for deleted line
;(OPPORTUNITY-LINE-NOT-DELETED)
(calc-penalty 'r3 'op3 'e32 9 'e31 3)
;(SECOND-OPPORTUNITY-CELL-NOT-DELETED FIRST-OPPORTUNITY-CELL-NOT-
  DELETED OPPORTUNITY-SECOND-LEAST OPPORTUNITY-FIRST-NEXTLEAST
  OPPORTUNITY-LINE-NOT-DELETED)
(next-cp 'r3 'c1 'op4 '(op3 op2 op1))
(calc-penalty 'c1 'op4 'e21 15 'e31 3)
;(SECOND-OPPORTUNITY-CELL-NOT-DELETED OPPORTUNITY-SECOND-LEAST)
(next-cp 'c1 'c2 'op5 '(op4 op3 op2 op1))
```

Figure 7.4. (Continued)

```
(calc-penalty 'c2 'op5 'e22 10 'e12 5)
(next-cp 'c2 'c3 'op6 '(op5 op4 op3 op2 op1))
(calc-penalty 'c3 'op6 'e23 12 'e13 6)
(select-line 'c1 'c3)
(select-cell 'c1 'e11 '(o6 o5 o4 o3 o2 o1))
(determine-allocation-amount 'c1 'e11 70)
(adjust-amount 'c1 70 'r1 120 'c1 'e11 70)
(delete-line 'c1 '(e11 e21 e31))
;-----------------------------------------------
(first-cp 'r1 'op1 nil)
(calc-penalty 'r1 'op1 'e13 6 'e12 5)
(next-cp 'r1 'r2 'op2 '(op1))
(calc-penalty 'r2 'op2 'e23 12 'e22 10)
(next-cp 'r2 'r3 'op3 '(op2 op1)) ; ;error - cal-penalty for deleted line
(calc-penalty 'r3 'op3 'e32 9 'e31 3)
(next-cp 'r3 'c1 'op4 '(op3 op2 op1)) ;error - cal-penalty for deleted line
(calc-penalty 'c1 'op4 'e21 15 'e31 3)
(next-cp 'c1 'c2 'op5 '(op4 op3 op2 op1))
(calc-penalty 'c2 'op5 'e22 10 'e12 5)
(next-cp 'c2 'c3 'op6 '(op5 op4 op3 op2 op1))
(calc-penalty 'c3 'op6 'e23 12 'e13 6)
(select-line 'c3 'c3)
(select-cell 'e13 '(o4 o3 o2 o1))
(determine-allocation-amount 'r1 'e13 50)
(adjust-amount 'r1 50 'c3 60 50 'c3 'e13 50)
(delete-line 'r1 '(e11 e12 e13))
;-----------------------------------------------
(copy-allocation 'c2 70 'e22 'r2 80)
(copy-allocation 'c3 10 'e23 'r2 10)
```

Figure 7.5. The action protocol of subject-3

From \ To	DEMAND 1	DEMAND 2	DEMAND 3	Supply	
SUPPLY 1	8 70	5	6	120	1
SUPPLY 2	15 ___	10	12 (60)	80	2
SUPPLY 3	3 80	9 ___	10 ___	80	
Demand	150	70	60	280	
		5	6		

(first-cp 'r1 'op1 nil)
(calc-penalty 'r1 'op1 'e13 6 'e12 5)
(next-cp 'r1 'r2 'op2 '(op1))
(calc-penalty 'r2 'op2 'e23 12 'e22 10)
(next-cp 'r2 'r3 'op3 '(op2 op1))
(calc-penalty 'r3 'op3 'e32 9 'e31 3)
(next-cp 'r3 'c1 'op4 '(op3 op2 op1))
(calc-penalty 'c1 'op4 'e11 8 'e31 3)
(next-cp 'c1 'c2 'op5 '(op4 op3 op2 op1))
(calc-penalty 'c2 'op5 'e32 9 'e12 5)
(next-cp 'c2 'c3 'op6 '(op5 op4 op3 op2 op1))
(calc-penalty 'c3 'op6 'e33 10 'e13 6)
(select-line 'r3 'c3)
(select-cell 'r3 'e31 '(op6 op5 op4 op3 op2 op1))
(determine-allocation-amount 'r3 'e31 80)
(adjust-amount 'r3 80 'c1 150 'r3 'e31 80)
(delete-line 'r3 '(e31 e32 e33))
;----------
(first-cp 'r1 'op1 nil)
(calc-penalty 'r1 'op1 'e13 6 'e12 5)
(next-cp 'r1 'r2 'op2 '(op1))
(calc-penalty 'r2 'op2 'e23 12 'e22 10)
(next-cp 'r2 'c1 'op3 '(op2 op1))
(calc-penalty 'c1 'op3 'e21 15 'e11 8)
(next-cp 'c1 'c2 'op4 '(op3 op2 op1))
(calc-penalty 'c2 'op4 'e22 10 'e12 5)
(next-cp 'c2 'c3 'op5 '(op4 op3 op2 op1))
(calc-penalty 'c3 'op5 'e23 12 'e13 6)
(select-line 'c1 'c3)
(select-cell 'c1 'e11 '(op5 op4 op3 op2 op1))
(determine-allocation-amount 'c1 'e11 70)
(adjust-amount 'c1 70 'r1 120 'c1 'e11 70)

Figure 7.5. (Continued)

```
(delete-line 'c1 '(e11 e21 e31))
;-----------------------------------------
(first-cp 'r1 'op1 nil)
(calc-penalty 'r1 'op1 'e13 6 'e12 5)
(next-cp 'r1 'r2 'op2 '(op1))
(calc-penalty 'r2 'op2 'e23 12 'e22 10)
(next-cp 'r2 'c2 'op3 '(op2 op1))
(calc-penalty 'c2 'op3 'e22 10 'e12 5)
(next-cp 'c2 'c3 'op4 '(op3 op2 op1))
(calc-penalty 'c3 'op4 'e23 12 'e13 6)
(select-line 'c3 'c3)
(select-cell 'c3 'e23 '(op4 op3 op2 op1))
;(SELECT-LEAST-COST-CELL)
(determine-allocation-amount 'c3 'e23 60)
(adjust-amount 'r1 50 'c3 60 'c3 'e23 60)
(delete-line 'c3 '(e11 e12 e13))
;-----------------------------------------
(copy-allocation 'c2 20 'e22 'r2 80)
(copy-allocation 'c3 60 'e23 'r2 60)
```

Figure 7.6. The action protocol of subject-4

From \ To	DEMAND 1	DEMAND 2	DEMAND 3	Supply
SUPPLY 1	8	5	6 (60)	120
SUPPLY 2	15 ___	10	12	80
SUPPLY 3	3 80	9 ___	10 ___	80
Demand	150	70	60	280

1

2

5 6

```
(first-cp 'r1 'op1 nil)
(calc-penalty 'r1 'op1 'e13 6 'e12 5)
(next-cp 'r1 'r2 'op2 '(op1))
(calc-penalty 'r2 'op2 'e23 12 'e22 10)
(next-cp 'r2 'r3 'op3 '(op2 op1))
(calc-penalty 'r3 'op3 'e32 9 'e31 3)
(next-cp 'r3 'c1 'op4 '(op3 op2 op1))
(calc-penalty 'c1 'op4 'e11 8 'e31 3)
(next-cp 'c1 'c2 'op5 '(op4 op3 op2 op1))
(calc-penalty 'c2 'op5 'e32 9 'e12 5)
(next-cp 'c2 'c3 'op6 '(op5 op4 op3 op2 op1))
(calc-penalty 'c3 'op6 'e33 10 'e13 6)
(select-line 'r3 'c3)
(select-cell 'r3 'e31 '(op6 op5 op4 op3 op2 op1))
(determine-allocation-amount 'r3 'e31 80)
(adjust-amount 'r3 80 'c1 150 'r3 'e31 80)
(delete-line 'r3 '(e31 e32 e33)
;-----------------------------------------
(first-cp 'r1 'op1 nil)
(calc-penalty 'r1 'op1 'e13 6 'e12 5)
(next-cp 'r1 'r2 'op2 '(op1))
(calc-penalty 'r2 'op2 'e23 12 'e22 10)
(next-cp 'r2 'c2 'op3 '(op2 op1))
(calc-penalty 'c2 'op3 'e22 10 'e12 5)
(next-cp 'c2 'c3 'op4 '(op3 op2 op1))
(calc-penalty 'c3 'op4 'e23 12 'e13 6)
(select-line 'c3 'c3)
;(SELECT-LINE-AFTER-ALL-OPPORTUNITIES)
(select-cell 'c3 'e13 '(op5 op4 op3 op2 op1))
(determine-allocation-amount 'c3 'e13 60)
(adjust-amount 'c3 60 'r1 120 'c1 'e13 60)
(delete-line 'c3 '(e13 e23 e33))
;-----------------------------------------
```

Figure 7.6. (Continued)

```
(first-cp 'r1 'op1 nil)
(calc-penalty 'r1 'op1 'e11 8 'e12 5)
(next-cp 'r1 'r2 'op2 '(op1))
(calc-penalty 'r2 'op2 'e21 15 'e22 10)
(next-cp 'r2 'c2 'op3 '(op2 op1))
(calc-penalty 'c2 'op3 'e22 10 'e12 5)
(next-cp 'c2 'c1 'op4 '(op3 op2 op1))
(calc-penalty 'c1 'op4 'e21 15 'e11 8)
(select-line 'c1 'c1)
(select-cell 'c1 'e11 '(op4 op3 op2 op1))
(determine-allocation-amount 'c1 'e11 60)
(adjust-amount 'r1 60 'c3 70 'c1 'e11 60)
;(VALUE=TOTAL+CURRENT)
(delete-line 'r1 '(e11 e12 e13))
;(VALUE=TOTAL+CURRENT)
;-----------------------------------------
(copy-allocation 'c1 10 'e21 'r2 80)
;(VALUE=TOTAL+CURRENT)
(copy-allocation 'c2 70 'e22 'r2 70)
;(VALUE=TOTAL+CURRENT)
```

Figure 7.7. The action protocol of subject-5

From \ To	DEMAND 1	DEMAND 2	DEMAND 3	Supply	
SUPPLY 1	8 70	5	6 (60)	120	1
SUPPLY 2	15 ___	10	12	80	2
SUPPLY 3	3 80	9 ___	10 ___	80	6
Demand	150	70	60	280	
		5	6		

```
(first-cp 'r1 'op1 nil)
(calc-penalty 'r1 'op1 'e13 6 'e12 5)
(next-cp 'r1 'r2 'op2 '(op1))
(calc-penalty 'r2 'op2 'e23 12 'e22 10)
(next-cp 'r2 'r3 'op3 '(op2 op1))
(calc-penalty 'r3 'op3 'e32 9 'e31 3)
(next-cp 'r3 'c1 'op4 '(op3 op2 op1))
(calc-penalty 'c1 'op4 'e11 8 'e31 3)
(next-cp 'c1 'c2 'op5 '(op4 op3 op2 op1))
(calc-penalty 'c2 'op5 'e32 9 'e12 5)
(next-cp 'c2 'c3 'op6 '(op5 op4 op3 op2 op1))
(calc-penalty 'c3 'op6 'e33 10 'e13 6)
(select-line 'r3 'c3)
(select-cell 'r3 'e31 '(op6 op5 op4 op3 op2 op1))
(determine-allocation-amount 'r3 'e31 80)
(adjust-amount 'r3 80 'c1 150 'r3 'e31 80)
(delete-line 'r3 '(e31 e32 e33))
;-----------------------------------
(first-cp 'r1 'op1 nil)
(calc-penalty 'r1 'op1 'e13 6 'e12 5)
(next-cp 'r1 'r2 'op2 '(op1))
(calc-penalty 'r2 'op2 'e23 12 'e22 10)
(next-cp 'r2 'r3 'op3 '(op2 op1))
;(OPPORTUNITY-LINE-NOT-DELETED)
(calc-penalty 'r3 'op3 'e32 9 'e31 3)
; (SECOND-OPPORTUNITY-CELL-NOT-DELETED FIRST-OPPORTUNITY-CELL-NOT-
   DELETED OPPORTUNITY-SECOND-LEAST OPPORTUNITY-FIRST-NEXTLEAST
   OPPORTUNITY-LINE-NOT-DELETED)
(next-cp 'r3 'c1 'op4 '(op3 op2 op1))
(calc-penalty 'c1 'op4 'e21 15 'e11 8)
(next-cp 'c1 'c2 'op5 '(op4 op3 op2 op1))
(calc-penalty 'c2 'op5 'e32 9 'e12 5)
;(FIRST-OPPORTUNITY-CELL-NOT-DELETED OPPORTUNITY-FIRST-NEXTLEAST)
```

Figure 7.7. (Continued)

```
(next-cp 'c2 'c3 'op6 '(op5 op4 op3 op2 op1))
(calc-penalty 'c3 'op6 'e33 10 'e13 6)
;(FIRST-OPPORTUNITY-CELL-NOT-DELETED OPPORTUNITY-FIRST-NEXTLEAST)
(select-line 'c1 'c3)
(select-cell 'c1 'e11 '(op6 op5 op4 op3 op2 op1))
(determine-allocation-amount 'c1 'e11 70)
(adjust-amount 'c1 70 'r1 120 'c1 'e11 70)
(delete-line 'c1 '(e11 e21 e31))
;-----------------------------------------
(first-cp 'r1 'op1 nil)
(calc-penalty 'r1 'op1 'e13 6 'e12 5)
(next-cp 'r1 'r2 'op2 '(op1))
(calc-penalty 'r2 'op2 'e23 12 'e22 10)
(next-cp 'r2 'c2 'op3 '(op2 op1))
(calc-penalty 'c2 'op3 'e32 9 'e12 5)
;(FIRST-OPPORTUNITY-CELL-NOT-DELETED OPPORTUNITY-FIRST-NEXTLEAST)
(next-cp 'c2 'c3 'op4 '(op3 op2 op1))
(calc-penalty 'c3 'op4 'e33 10 'e13 6)
(select-line 'c2 'c3)
(select-cell 'c2 'e12 '(op4 op3 op2 op1))
(determine-allocation-amount 'c2 'e12 50)
(adjust-amount 'r1 50 'c3 60 'c2 'e12 50)
;(VALUE=TOTAL+CURRENT)
(delete-line 'r1 '(e11 e12 e13))
;(VALUE=TOTAL+CURRENT)
;-----------------------------------------
(first-cp 'r2 'op1 nil)
;(PREFER-COPY-OVER-CALC-PENALTY VALUE=TOTAL+CURRENT)
(calc-penalty 'r2 'op1 'e23 12 'e22 10)
;(PREFER-COPY-OVER-CALC-PENALTY VALUE=TOTAL+CURRENT)
(next-cp 'r2 'r3 'op2 '(op1))
;(PREFER-COPY-OVER-CALC-PENALTY VALUE=TOTAL+CURRENT OPPORTUNITY-LINE-
NOT-DELETED)
(calc-penalty 'r3 'op2 'e33 10 'e32 9)
;(PREFER-COPY-OVER-CALC-PENALTY VALUE=TOTAL+CURRENT SECOND-
  OPPORTUNITY-CELL-NOT DELETED FIRST-OPPORTUNITY-CELL-NOT-DELETED
  OPPORTUNITY-SECOND-LEAST OPPORTUNITY-FIRST-NEXTLEAST OPPORTUNITY-LINE-
  NOT-DELETED)
(next-cp 'r3 'c2 'op3 '(op2 op1))
;(PREFER-COPY-OVER-CALC-PENALTY VALUE=TOTAL+CURRENT)
(calc-penalty 'c2 'op3 'e22 10 'e32 9)
;(PREFER-COPY-OVER-CALC-PENALTY VALUE=TOTAL+CURRENT SECOND-
  OPPORTUNITY-CELL-NOT-DELETED OPPORTUNITY-SECOND-LEAST OPPORTUNITY-
  FIRST-NEXTLEAST)
(next-cp 'c2 'c3 'op4 '(op3 op2 op1))
;(PREFER-COPY-OVER-CALC-PENALTY VALUE=TOTAL+CURRENT)
(calc-penalty 'c3 'op4 'e33 10 'e13 6)
;(PREFER-COPY-OVER-CALC-PENALTY VALUE=TOTAL+CURRENT SECOND-
  OPPORTUNITY-CELL-NOT-DELETED FIRST-OPPORTUNITY-CELL-NOT-DELETED
  OPPORTUNITY-SECOND-LEAST OPPORTUNITY-FIRST-NEXTLEAST)
(select-line 'c3 'c3)
;(VALUE=TOTAL+CURRENT)
(select-cell 'c3 'e13 '(op4 op3 op2 op1))
;(VALUE=TOTAL+CURRENT SELECT-LEAST-COST-CELL)
(determine-allocation-amount 'c3 'e13 50)
```

Figure 7.7. (Continued)

```
;(VALUE=TOTAL+CURRENT)
(adjust-amount 'r1 50 'c3 60 'c3 'e13 50)
;(VALUE=TOTAL+CURRENT DELETE-LINE-WHEN-ALL-AMOUNT-ALLOCATED)
;-----------------------------------
(copy-allocation 'c2 20 'e22)
;(VALUE=TOTAL+CURRENT DELETE-LINE-WHEN-ALL-AMOUNT-ALLOCATED)
(copy-allocation 'c3 10 'e23)
;(VALUE=TOTAL+CURRENT DELETE-LINE-WHEN-ALL-AMOUNT-ALLOCATED)
```

Chapter 8 LIMITATIONS, IMPLICATIONS, AND FUTURE DIRECTIONS

8.1 Limitations

A factor that may prohibit PSCD from practical application is its inefficiency. The biggest bottleneck may be the pattern matcher. In the performance mode, PSCD must match all productions to working memory and find all instantiations of applicable productions, which take some time. In addition PSCD must find violated constraints for every states generated by possible instantiations. This process actually takes a lot of time. For example, suppose PSCD finds x number of instantiations and has y number of constraints, the matcher has to be called y^x times! One way of avoiding this situation may be the use of structured object representation such as frames or scripts for the representation of working memory elements.

In validating PSCD, this study used a limited set of problems and a small set of students. It was difficult for us to have a large amount of data because PSCD is not a fully implemented ITS. When PSCD becomes fully implemented, PSCD can collect error data when students use PSCD. Since such error data are in machine readable forms, they can be easily transcribed to action protocols by a simple function. Implementing PSCD fully is the next research project.

From the experience of building PSCD, this researcher became strongly to agree with VanLehn (1983), that getting the right representation language is the most important aspect of building a computational model. A number of different representation languages were considered for modeling the task of transportation problem solving before the current representation language was determined, but the current language still needs more improvement. With the

current language, it was a little difficult to separate relevance conditions and satisfaction conditions in some cases. For example, we expressed the constraint 14 that says "Select a line after all undeleted lines are associated with some opportunity" as ((inst =line-1 line)(allocate =line-1)(inst =line line)(status =line nil)(*no (relate =some =line))** nil)). The exact interpretation of this expressions would be that If there is a line "line-1" that is selected for allocation, and there is another "line" whose state is not deleted and the "line" is not related with any opportunity, then the state violates this constraint. Although this expression achieves its goal in representing its original concept, it blurs the distinction between the relevance and the satisfaction conditions. More study must be directed to devise a better representation language.

8.2 Implications and Future Directions

This study implies that integration of theories is possible with the computational modeling approach and also fruitful in that it enables practical applications. Experience from this study also suggests that it may be possible to integrate even the normative theory, i.e., the theory of rational choice. For example, the control problem of a production system (i.e., conflict resolution) can be viewed as selecting the most promising state out of possible states. It appears that this integration with the normative theory would realize more practical decision aids in business domains and also provide ample opportunities for exploring alternative decision theories.

The PSCD architecture proposed in this study has some limitations as stated in the previous section. It appears that these limitations are not major disadvantages, but related to minor implementation details. Inefficiency is pointed out as a problem for inhibiting practical application. The methodology of

object-oriented programming appears to be useful in alleviating the inefficiency problem of PSCD. It is uncertain how much the methodology can contribute, but in general structured object representations are more efficient because related facts can be retrieved easily. However, structuring of related objects will reduce the flexibility of reasoning. Apparently more study is needed.

A promising path for future studies is to integrate a planning technique with a learning model, as VanLehn (1987) did. In fact PSCD assumes that all observable, physical activity is available. Thus when designing an interface, it is necessary to have all primitive actions present in the screen and to require students' to choose an action after an action. Only in this way can PSCD have all observable information necessary for diagnosis. If a planning technique were incorporated in PSCD, interface design would be simpler and nicer because a planning technique would allow PSCD to infer missing actions.

Yet another promising path is to explore the relationship between explanation-based learning and state constraints theory, and between empirical and rational learning. Both explanation-based learning and state constraint theory are rational learning, but they are different in that explanation-based learning works on state generation whereas state constraints theory works on state evaluations. Integration of these two types of rational learning mechanisms would yield a better computational model. Integrated learning which attempts to integrate empirical and rational learning is a recent topic that attracts much attention (Pazzani, 1990). Integration in itself is worth try.

A recent study shows that semantic construction is not limited to the use of principled knowledge that are already possessed by learners, but can lead to development of new principled knowledge (Chi and VanLehn, 1991). This is the question how you get principled knowledge in the first place. Chi and

VanLehn (1991) called the process of creating new knowledge as "construction." They posited several subprocesses underlying construction, such as a natural language inferencing using common sense knowledge, inferencing based on new information presented in the example, and a kind of generalization after several self-explanations have been generated. This is another area to explore.

Finally, but not the least important, much efforts must be directed to integrate cognitive theories with management related theories. For example, it may be possible to consider possible effects of task characteristics on information processing, known as the contingent decision hypothesis. The contingent decision hypothesis states that judgment and choice are highly contingent on task characteristics (Einhorn and Hogarth, 1981). Further it is argued that an individual utilizes a number of different information-processing strategies to solve decision tasks, and any decision strategy has certain benefits (accuracy) associated with its use and also certain costs (efforts) (Johnson and Payne, 1985; Johnson et al., 1988; Payne, 1976; Payne et al. 1988; Payne et al., 1990). Decision rule selection would then consider both the costs and benefits associated with each possible strategy, and select the best strategy. The idea of the strategy selection by the trade-off between strategy's effort and strategy's accuracy was also supported by several authors (Beach and Mitchell,1978) who argued strongly for the idea that strategy selection is the result of a compromise between the desire to make a correct decision and the desire to minimize effort. One way to incorporate the theory to PSCD is to represent decision strategies as rules, use a search mechanism for guiding rule firings, and have the search being directed by an evaluation function which represents the trade-off relationship between accuracy and efforts.

Another way to incorporate the contingent decision hypothesis in the current PSCD architecture may be to posit a meta-cognitive mechanism which guides the selection of an information processing strategies from a number of alternatives (e.g., elimination by aspects, satisficing, lexicographic choice, equal weighting), and use the selected strategy for search through the problem space. Although this appears to be a better way, there is a problem of validation. Such a study may need more dense protocol to validate a meta-cognitive mechanism, which means more time and efforts. This topic is set aside for future research. It is believed that the "task" variable is important because it provides a bridge between theories at the individual level (e.g.., organizational behavior or HIP) and theories at the organizational level (e.g., organizational theories). In recent years, there have been many attempts to incorporate organizational context variables into the design of decision aids. This literature is often called *computer-supported cooperative work* (Greif, 1988). In later works, this researcher will be directing more efforts to the integration of cognitive theories with high level theories employed in business studies.

Bibliography

Alter, S. L. Decision Support Systems: Current Practice and Continuing Challenge, Reading, MA: Addison-Wesley Publishing Company, 1980.

Anderson, J. R. Language, Memory, and Thought, Hillsdale, NJ: Lawrence Erlbaum Associates, 1976.

Anderson, J. R., Greeno, J. G., Kline, P. J., and Neves, D. M. "Acquisition of Problem Solving Skill," In Cognitive Skills and Their Acquisition, Ed. Anderson, J. R., Hillsdale, NJ: Lawrence Erlbaum Associates, 1981, pp. 191-230.

Anderson, J. A. The Architecture of Cognition, Cambridge, MA: Harvard University Press, 1983.

Anderson, J. R., Farrell, R., and Sauers, R. "Learning to Program in LISP," Cognitive Science 8, 1984, pp. 87-129.

Anderson, J. R. and Reiser, B. J. "The LISP Tutor," Byte 10, 1985, pp. 159-175.

Anderson, J. R. and Jeffries, R. "Novice LISP Errors: Undetected Losses of Information from Working Memory," Human-Computer Interaction 1, 1985, pp. 107-131.

Anderson, J. R., Boyle, C. F., and Reiser, B. J. "Intelligent Tutoring Systems," Science 228(4698), 1985, pp. 456-462.

Anderson, J. R. "Methodologies for Studying Human Knowledge," Behavioral and Brain Sciences 10, 1987, pp. 467-505.

Anderson, J. R., Corbett, A. T., and Reiser, B. J. Essential LISP, Reading, MA: Addison-Wesley Publishing Company, Inc., 1987.

Anderson, J. R. "The Expert Module," In Foundations of Intelligent Tutoring Systems, Ed. Polson, M. C. and Richardson, J. J., Hillsdale, NY: Lawrence Erlbaum, 1988, pp. 21-53.

Anderson, J. R. and Thompson, R. "Use of Analogy in a Production System Architecture," In Similarity and Analogical Reasoning, Ed. Vosniadou, S. and Ortony, A., New York, NY: Cambridge University Press, 1989, pp. 267-297.

Anderson, J. R. "A Theory of the Origins of Human Knowledge," Artificial Intelligence 40, 1989, pp. 313-351.

Anderson, J. A. The Adaptive Character of Thought, Hillsdale, NY: Lawrence Erlbaum Associates, 1990.

Anderson, J. R., Boyle, C. F., Corbett, A. T., and Lewis, M. W. "Cognitive Modeling and Debugging Novice Programs," Artificial Intelligence 42, 1990, pp. 7-50.

Anzai, Y. and Simon, H. A. "The Theory of Learning by Doing," Psychological Review 86, 1979, pp. 124-140.

Argyris, C. "Organizational Learning and Management Information Systems," Accounting, Organization and Society 2(2), 1977, pp. 113-123.

Ashton, R. H. "An Experimental Study of Internal Control Judgments," Journal of Accounting Research 12, 1974, pp. 143-157.

Bailey, J., A. D., Hackenbrack, K., De, P., and Dillard, J. "Artificial Intelligence, Cognitive Science, and Computational Modeling in Auditing Research: A Research Approach," Journal of Information Systems (Spring), 1987, pp. 20-40.

Baroody, A. J. and Ginsburg, H. P. "The Relationship between Initial Meaningful and Mechanical Knowledge of Arithmetic," In Conceptual and Procedural Knowledge: The Case of Mathematics, Ed. Hiebert, J., Hillsdale, NJ: Lawrence Erlbaum Associates, 1986, pp. 75-107.

Barr, A. and Feigenbaum, E. A. The Handbook of Artificial Intelligence, Los Altos, CA: William Kaufmann, Inc., 1981.

Beach, L. R. and Mitchell, T. R. "A Contingency Model for the Selection of Decision Strategies," Academy of Management Review 3, 1978, pp. 439-449.

Bhaskar, R. and Dillard, J. F. "Human Cognition in Accounting: A Preliminary Analysis," In Behavioral Experiments in Accounting II, Ed. Burns, T. J., Columbus, OH: Ohio State University, 1979, pp. 323-385.

Blanning, R. W. (ed.) Foundations of Expert Systems for Management, Koln, Germany: Verlag, 1990.

Bower, G. H. and Clapper, J. P. "Experimental Methods in Cognitive Science," In Foundations of Cognitive Science, Ed. Posner, M. I., Cambridge, MA: MIT press, 1989, pp. 245-230.

Brachman, R. J. and Levesque, H. J. (ed.) Readings in Knowledge Representation, Sam Mateo, CA: Morgan Kaufmann Publishers, Inc., 1985.

Briars, D. J. and Larkin, J. H. "An Integrated Model of Skills in Solving Elementary Word Problems," Cognition and Instruction 20, 1984, pp. 245-296.

Brown, J. S. and Burton, R. "Diagnostic Models for Procedural Bugs in Basic Mathematical Skills," Cognitive Science 2, 1978, pp. 155-192.

Brown, J. S. and VanLehn, K. "Repair Theory: A Generative Theory of Bugs in Procedural Skills," Cognitive Science 4, 1980, pp. 379 - 426.

Brown, J. S. and VanLehn, K. "Toward a Generative Theory of "bugs"," In Addition and Subtraction: A Cognitive Perspective, Ed. Carpenter, T. P. and Romberg, T. A., Hillsdale, NJ: Lawrence Erlbaum Associates, 1982, pp. 117-135.

Brownston, L., Farrell, R., Kant, E., and Martin, N. Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming, Reading, MA: Addison-Wesley, 1985.

Buchanan, B. G. and Shortliffe, E. H. (ed.) Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Reading, MA: Addison-Wesley, 1984.

Bundy, A. and Silver, B. "A Critical Survey of Rule Learning Programs," In Proceedings of the European Conference on Artificial Intelligence, Orsay, France: 1982, pp. 151-157.

Bundy, A., Silver, B., and Plummer, D. "An Analytical Comparison of Some Rule-Learning Programs," Artificial Intelligence 27, 1985, pp. 137-181.

Burns, H., Parlett, J. W., and Redfield, C. L. (ed.) Intelligent Tutoring Systems: Evolutions in Design, Hillsdale, NJ: Lawrence Erlbaum Associates, 1991.

Burton, R. R. "Diagnosing Bugs in a Simple Procedure Skill," In Intelligent Tutoring Systems, Ed. Sleeman, D. and Brown, J. S., New York, NY: Academic Press, 1982, pp. 79-98.

Caplan, E. H. "Behavioral Accounting Research - An Overview," In Behavioral Accounting Research: A Critical Analysis, Ed. Ferris, K. R., Columbus, OH: Century VII Publishing Company, 1988, pp. 3-12.

Card, S. K., Moran, T. P., and Newell, A. The Psychology of Human-Computer Interaction, Hillsdale, NJ: Lawrence Erlbaum Associates, 1983.

Charniak, E. and McDermott, D. Introduction to Artificial Intelligence, Reading, MA: Addison-Wesley, 1985.

Chi, M. T. H. and VanLehn, K. A. "The Content of Physics Self-Explanations," The Journal of the Learning Sciences 1(1), 1991, pp. 69-105.

Chomsky, N. Aspects of the Theory of Syntax, Cambridge, MA: The MIT Press, 1965.

Clancey, W. J. "The Epistemology of a Rule-Based Expert-Systems: A Framework for Explanation," Artificial Intelligence 20, 1983, pp. 215-251.

Clancey, W. J. "Methodology for Building an Intelligent Tutoring System," In Methods and Tactics in Cognitive Science, Ed. Kintsch, W., Polson, P. G., and Miller, J. R., Hillsdale, NJ: Lawrence Erlbaum Associates, 1984, pp. 51 - 83.

Clancey, W. J. "From GUIDON to NEOMYCIN and HERACLES in Twenty Short Lessons: ONR Final Report 1979-1985," AI Magazine 7(3), 1986, pp. 40-60.

Clancey, W. J. "Intelligent Tutoring Systems: A Tutorial Survey," In Current Issues in Expert Systems, Academic Press, 1987a, pp. 38 - 78.

Clancey, W. J. Knowledge-Based Tutoring, Cambridge, MA: The MIT Press, 1987b.

Clancey, W. J. and Bock, B. "Representing Control Knowledge as Abstract Tasks and Metarules," In Expert System Applications, Ed. Bock, C. and Coombs, M. J., Berlin, Germany: Springer-Verlag, 1988, pp. 1-77.

Clancey, W. J. "Artificial Intelligence and Learning Environments: Preface," Artificial Intelligence 42(1), 1990, pp. 1 - 6.

Cohen, P. R. and Feigenbaum, E. A. (ed.) The Handbook of Artificial Intelligence, Los Altos, CA: William Kaufman, 1982.

Davis, R., Buchanan, B., and Shortliffe, E. "Production Rules as a Representation for a Knowledge-Based Consultation Program," Artificial Intelligence 8(1), 1977, pp. 15-45.

Davis, R. and Buchanan, B. G. "Meta-Level Knowledge," In Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Ed. Buchanan, B. G., Reading, MA: Addison-Wesley, 1984, pp. 507-530.

Davis, G. B. and Olson, M. H. Management Information Systems: Conceptual Foundations, Structure, and Development, New York, NY: McGraw-Hill Book Company, 1985.

Dawes, R. M. and Corrigan, B. "Linear Models in Decision-making," Psychological Bulletin 81, 1974, pp. 95-106.

Dejong, G. "An Introduction to Explanation-based Learning," In Exploring Artificial Intelligence, Ed. Shrobe, H. E., San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1988, pp. 45-82.

Demski, J. S. Information Analysis, Reading, MA: Addison Wesley, 1972.

Dillard, J. F. "Cognitive Science and Decision-making Research in Accounting," Accounting, Organizations and Society 9(3/4), 1984, pp. 343-354.

Draper, S. W. "Machine Learning and Cognitive Development," In Computers, Cognition and Development, Ed. Rutkowska, J. and Crook, C., New York, NY: John Wiley & Sons Ltd, 1987, pp. 255-1987.

Ein-Dor, P. and Segev, E. Managing Management Information Systems, Lexington Books, 1977.

Einhorn, H. J. and Hogarth, R. M. "Behavioral Decision Theory: Processes Judgement of Choice," Annual Review of Psychology 32, 1981, pp. 52 - 88.

Forsyth, R. and Rada, R. Machine Learning: Applications in Expert Systems and Information Retrieval, New York, NY: John Wiley & Sons, 1986.

Frost, R. Introduction to Knowledge Base Systems, New York, NY: MacMillan Publishing Company, 1986.

Gelman, R. and Gallistel, C. R. The Child's Understanding of Number, Cambridge, MA: Harvard University Press, 1978.

Gelman, R. and Meck, E. "Preschoolers' Counting: Principle before Skill," Cognition 13, 1983, pp. 343-359.

Gelman, R. and Meck, E. "The Notion of Principle: The Case of Counting," In Conceptual and Procedural Knowledge: The Case of Mathematics, Ed. Hiebert, J., Hillsdale, NJ: Lawrence Erlbaum Associates, 1986, pp. 29-55.

Gentner, D. "The Mechanisms of Analogical Learning," In Similarity and Analogical Reasoning, Ed. Vosniadou, S. and Ortony, A., New York, NY: Cambridge University Press, 1989, pp. 199-241.

Georgeff, M. P. "Procedural Control in Production Systems," Artificial Intelligence 18, 1982, pp. 175-201.

Gick, M. L. and Holyoak, K. J. "Analogical Problem Solving," Cognitive Psychology 12, 1980, pp. 306-355.

Gick, M. L. and Holyoak, K. J. "Schema Induction and Analogical Transfer," Cognitive Psychology 15, 1983, pp. 1-38.

Glaser, R. "Special Issue: Psychological Science and Education," American Psychologists 40(10), 1986, pp. 573-574.

Glaser, R. and Bassok, M. "Learning Theory and the Study of Instruction," Annual Review of Psychology 40, 1989, pp. 631-666.

Glaser, R. "The Reemergence of Learning Theory within Instructional Research," American Psychologist 45(1), 1990, pp. 29-39.

Gorry, G. A. and Scott Morton, M. S. "A Framework for Management Information Systems," Sloan Management Review 13(1), 1971, pp. 55-70.

Green, P. E. and Rao, V. R. Applied Multidimensional Scaling: A Comparison of Approaches and Algorithms, New York, NY: Holt, Rinehart & Winston, 1972.

Greeno, J. G., Riley, M. S., and Gerlman, R. "Conceptual Competence for Children's Counting," Cognitive Psychology 16, 1984, pp. 94-143.

Greeno, J. G. "Situations, Mental Models, and Generative Knowledge," In Complex Information Processing: The Impact of Herbert A. Simon, Ed. Klahr, D. and Kotovsky, K., Hillsdale, NJ: Lawrence Erlbaum Associates, 1989, pp. 285-318.

Greif, I. (ed.) Computer-Supported Cooperative Work: A Book of Readings, San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1988.

Hayes, J. R. and Simon, H. A. "Psychological Differences among Problem Isomorphs," In Cognitive Theory, Ed. Castellan, J., Pisoni, D. B., and Potts, G., Hillsdale, NJ: Lawrence Erlbaum Associates, 1974, pp. 21-41.

Hayes-Roth, F. and McDermott, J. "Learning Structured Patterns from Examples," In Proceedings of Third International Joint Conference on Pattern Recognition, 1976, pp. 419-423.

Hiebert, J. (ed.) Conceptual and Procedural Knowledge: The Case of Mathematics, Hillsdale, NY: Lawrence Erlbaum Associates, 1986.

Hiebert, J. and Lefevre, P. "Conceptual and Procedural Knowledge in Mathematics," In Conceptual and Procedural Knowledge: The Case of Mathematics, Ed. Hiebert, J., Hillsdale, NJ: Lawrence Erlbaum Associates, 1986, pp. 1-23.

Hoffman, P. J., Slovic, P., and Rorer, L. G. "An Analysis-of-Variance Model for the Assessment of Configural Cue Utilization in Clinical Judgement," Psychological Bulletin 69, 1968, pp. 338-349.

Hogarth, R. M. Judgment and Choice: The Psychology of Decision, Chichester, England: Wiley, 1980.

Hogarth, R. M. and Reder, M. W. (ed.) Rational Choice: The Contrast between Economics and Psychology, Chicago, IL: The University of Chicago Press, 1987.

Hogarth, R. M. (ed.) Insights in Decision-making: A Tribute to Hillel J. Einhorn, Chicago, IL: The University of Chicago Press, 1990.

Holland, J. H., Holyoak, K. J., Nisbett, R. E., and Thagard, P. R. Induction: Processes of Inference, Learning, and Discovery, Cambridge, MA: MIT press, 1987.

Holyoak, K. J. and Thagard, P. R. "A Computational Model of Analogical Problem Solving," In Similarity and Analogical Reasoning, Ed. Vosniadou, S. and Ortony, A., New York, NY: Cambridge University Press, 1989, pp. 242-266.

Hursch, C., Hammond, K. R., and Hursch, J. L. "Some Methodological Considerations in Multiple Cue Probability Studies," Psychological Reviews 71, 1964, pp. 42-60.

Jackson, P. Introduction to Expert Systems, Reading, MA: Addison-Wesley, 1990.

Janis, I. L. and Mann, L. Decision-making: A Psychological Analysis of Conflict, Choice, and Commitment, New York, NY: Free Press, 1977.

Johnson, E. J. and Payne, J. W. "Effort and Accuracy in Choice," Management Science 31(4), 1985, pp. 395 - 414.

Johnson, W. L. Intention-Based Diagnosis of Novice Programming Errors, Los Altos, CA: Morgan Kaufmann Publishers, 1986.

Johnson, E. J., Payne, W., and Bettman, J. R. "Information Displays and Preference Reversals," Organizational Behavior and Human Decision Processes 42, 1988, pp. 1-21.

Johnson-Laird, P. N. Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness, New York, NY: Cambridge University Press, 1983.

Johnson-Laird, P. N. and Byrne, R. M. J. Deduction, Hillsdale, NJ: Lawrence Erlbaum Associates, 1990.

Joyce, E. J. and Biddle, G. C. "Anchoring and Adjustment in Probabilistic Inference in Auditing," Journal of Accounting Research, 1981, pp. 120-145.

Just, M. A. and Carpenter, P. A. "A Theory of Reading: From Eye Fixation to Comprehension," Psychological Review 87, 1980, pp. 329-354.

Kahneman, D., Slovic, P., and Tversky, A. Judgement under Uncertainty: Heuristics and Biases, Cambridge, MA: Cambridge University Press, 1982.

Kearsley, G. "Overview," In Artificial Intelligence and Instruction: Applications and Methods, Ed. Kearsley, G., Reading, MA: Addison-Wesley Publishing Company, 1987, pp. 3-10.

Keen, P. G. W. and Scott Morton, M. Decision Support Systems: An Organizational Perspective, Reading, MA: Addison Wesley, 1978.

Keeney, R. L. and Raiffa, H. Decisions with Multiple Objectives: Preferences and Value Tradeoffs, New York, NY: Wiley, 1976.

Kibler, D. and Langley, P. "Machine Learning as an Experimental Science," In Proceedings of the Third European Working Session on Learning, Ed. Sleeman, D., London, England: Pitman Publishing, 1988, pp. 81-91.

Kieras, D. E. "The Why, When, and How of Cognitive Simulation: A Tutorial," Behavioral Research Methods, Instruments, and Computers 22, 1985, pp. 265-394.

Kintsch, W., James, R. M., and Polson, P. G. (ed.) Method and Tactics in Cognitive Science, Hillsdale, NJ: Lawrence Erlbaum Associates, Inc., 1984.

Klahr, D. and Wallace, J. G. Cognitive Development: An Information Processing View, Hillsdale, NJ: Lawrence Erlbaum Associates, 1976.

Klahr, D., Langley, P., and Neches, R. (ed.) Production System Models of Learning and Development, Cambridge, MA: The MIT Press, 1987.

Ko, C. and Mock, T. J. "Behavioral Research in Accounting Information Systems," In Behavioral Accounting Research: A Critical Analysis, Ed. Ferris, K. R., Columbus, OH: Century VII Publishing Company, 1988, pp. 171-202.

Kodratoff, Y. Introduction to Machine Learning, Palo Alto, CA: Morgan Kauffman Publishers, 1986.

Kodratoff, Y. and Michalski, R. S. (ed.) Machine Learning: An Artificial Intelligence Approach, San Mateo, CA: Morgan Kaufman, 1990.

Kotovsky, K., Hayes, J. R., and Simon, H. A. "Why are Some Problems Hard? Evidence from Tower of Hanoi," Cognitive Psychology 17, 1985, pp. 248-294.

Kowalski, B. and VanLehn, K. "CIRRUS: Inducing Subject Models from Protocol Data," In The Tenth Annual Conference of the Cognitive Science Society, Ed. Patel, V. L. and Groen, G. J., Hillsdale, NJ: Lawrence Erlbaum Associates, 1988, pp. 623-629.

Kreps, D. M. Notes on the Theory of Choice, Boulder, CL: Westview Press, 1988.

Laird, J. E., Newell, A., and Rosenbloom, P. S. "SOAR: An Architecture for General Intelligence," Artificial Intelligence 33, 1987, pp. 1-64.

Langley, P., Neches, R., Neves, D., and Anzai, Y. "A Domain Independent Framework for Learning Procedures," International Journal of Policy Analysis and Information Systems 4(2), 1980, pp. 163-197.

Langley, P. and Simon, H. A. "The Central Role of Learning in Cognition," In Cognitive Skills and Their Acquisition, Ed. Anderson, J. R., Hillsdale, NJ: Lawrence Erlbaum Associates, 1981, pp. 361-380.

Langley, P. "Learning Search Strategies through Discrimination," International Journal of Man-Machine Studies 18, 1983a, pp. 513-541.

Langley, P. "Exploring the Space of Cognitive Architectures," Behavior Research Methods & Instrumentation 15(2), 1983b, pp. 289-299.

Langley, P. and Ohlsson, S. "Automated Cognitive Modeling," In Proceedings of the National Conference on Artificial Intelligence, Austin, TX: William Kaufmann, 1984, pp. 193-197.

Langley, P. "Learning to Search: From Weak Methods to Domain-Specific Heuristics," Cognitive Science 9, 1985, pp. 217-260.

Langley, P. "A General Theory of Discrimination Learning," In Production System Models of Learning and Development, Ed. Klahr, D., Langley, P., and Neches, R., Cambridge, MA: The MIT Press, 1987, pp. 99 - 162.

Langley, P., Wogulis, J., and Ohlsson, S. "Rules and Principles in Cognitive Diagnosis," In Diagnostic Monitoring of Skill and Knowledge Acquisition, Ed. Frederiksen, N., Hillsdale, NJ: Lawrence Erlbaum Associates, 1990, pp. 217-250.

Larkin, J. H., McDermott, J., Simon, D. P., and Simon, H. A. "Expert and Novice Performance in Solving Physics Problems," Science 208, 1980, pp. 1335-1342.

Larkin, J. H. "Enriching Formal Knowledge: A Model for Learning to Solve Problems in Physics," In Cognitive Skills and Their Acquisition, Ed. Anderson, J. R., Hillsdale, NJ: Lawrence Erlbaum Associates, 1981, pp. 311-344.

Larkin, J. H. "The Role of Problem Representation in Physics," In Mental Models, Ed. Gentner, D. and Stevens, A. L., Hillsdale, NJ: Lawrence Erlbaum Associates, 1983, pp. 75-98.

Larkin, J. H., McDermott, J., Simon, D. P., and Simon, H. A. "Models of Competence in Solving Physics Problems," Cognitive Science 4, 1984, pp. 317-345.

Larkin, J. H. and Simon, H. A. "Why a Diagram is (Sometimes) Worth 10,000 Words," Cognitive Science 11, 1987, pp. 65-100.

Larkin, J. H. "Display-Based Problem Solving," In Complex Information Processing: The Impact of Herbert A. Simon, Ed. Klahr, D. and Kotovsky, K., Hillsdale, NJ: Lawrence Erlbaum Associates, 1989, pp. 319-341.

Lee, S. M. Goal Programming for Decision Analysis, Philadelphia, PA: Auerbach Publishers, 1972.

Lee, S. M., Moore, L. J., and Taylor, B. W. Management Science, Needham Heights, MA: Allyn and Bacon, 1990.

Libby, R. "Accounting Ratios and the Prediction of Failure: Some Behavioral Evidence," Journal of Accounting Research 13, 1975, pp. 150-161.

Libby, R. "Bankers' and Auditors' Perceptions of the Message Communicated by the Audit Report," Journal of Accounting Research 17, 1979, pp. 99-122.

Libby, R. Accounting and HIP: Theory and Applications, Englewood Cliffs, NJ: Prentice-Hall, 1981.

Mandl, H. and Lesgold, A. (ed.) Learning Issues for Intelligent Tutoring Systems, New York, NY: Springer-Verlag, 1988.

Matz, M. "Toward A Process Model for High School Algebra Errors," In Intelligent Tutoring Systems, Ed. Sleeman, D. and Brown, J. S., New York, NY: Academic, 1982, pp. 25-49.

McDermott, J. and Forgy, C. "Production System Conflict Resolution Strategies," In Pattern-Directed Systems, Ed. Waterman, D. A. and Hayes-Roth, F., New York, NY: Academic Press, 1978, pp. 177-202.

McDermott, J. and Larkin, J. H. "Re-Representing Textbook Physics Problems," In Proceedings of the 2nd National Conference of the Canadian Society for Computational Studies of Intelligence, Toronto, Canada: University of Toronto Press, 1978, pp. 156-164.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (ed.) Machine Learning: An Artificial Intelligence Approach, Palo Alto, CA: Tioga, 1983.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (ed.) Machine Learning: An Artificial Intelligence Approach, Palo Altos, CA: Morgan Kaufman, 1986.

Minsky, M. "A Framework for representing Knowledge," In The Psychology of Computer Vision, Ed. Winston, P., New York, NY: McGraw-Hill, 1975, pp. 211-277.

Minton, S. Learning Search Control Knowledge: An Explanation-Based Approach, Norwell, MA: Kluwer Academic Publishers, 1988.

Mitchell, T. M. "Version Spaces: A Candidate Elimination Approach to Rule Learning," In Proceedings of the Fifth International Joint Conference on Artificial Intelligence, 1977, pp. 305-310.

Mitchell, T. M., Carbonell, J. G., and Michalski, R. S. (ed.) Machine Learning: A Guide to Current Research, Norwell, MA: Kluwer Academic Publishers, 1986.

Neches, R. T. "Simulation Systems for Cognitive Psychology," Behavioral Research Methods, Instruments, and Computers 14, 1982, pp. 77-91.

Neches, P., Langley, P., and Klahr, D. "Learning, Development, and Production Systems," In Production System Models of Learning and Development, Ed. Klahr, D., Langley, P., and Neches, R., Cambridge, MA: The MIT Press, 1987, pp. 1-53.

Newell, A. "Heuristic Programming: Ill-Structured Problems," In Progress in Operations Research, Ed. Aronofsky, J., 1969, pp. 362-414.

Newell, A. and Simon, H. A. Human Problem Solving, Englewood Cliffs, NJ: Prentice Hall, 1972.

Newell, A. "You Can't Play 20-Questions with Nature and Win: Projective Comments on the Papers of the Symposium," In Visual Information Processing, Ed. Chase, W. G., New York, NY: Academic Press, 1973a, pp. 283-308.

Newell, A. "Production Systems: Models of Control Structure," In Visual Information Processing, Ed. Chase, W., New York, NY: Academic Press, 1973b, pp. 463-526.

Newell, A. and Simon, H. A. "Computer Science as Empirical Inquiry: Symbols and Search," Communication of ACM 19, 1976, pp. 113-126.

Newell, A. "On the Analysis of Human Problem Solving Protocols," In Thinking: Readings in Cognitive Science, Ed. Johnson-Laird, P. N. and Wason, P. C., Cambridge, MA: Cambridge University Press, 1977, pp. 46-61.

Newell, A. "Reasoning, Problem Solving, and Decision Processes: The Problem Space as a Fundamental Category," In Attention and Performance, Ed. Nickerson, R., Hillsdale, NJ: Lawrence Erlbaum Associates, 1980, pp. 693-720.

Newell, A. "The Knowledge Level," The AI Magazine 18, 1981, pp. 1-20.

Newell, A. Unified Theories of Cognition, Cambridge, MA: Harvard University Press, 1990.

Nisbett, R. E. and Wilson, T. D. "Telling More Than We Can Know: Verbal Reports on Mental Processes," Psychological Review 84, 1977, pp. 231-259.

Norman, D. A. "Categorization of Action Slips," Psychological Review 88, 1981, pp. 1-15.

O'Leary, D. F. "Validation of Expert Systems-With Application to Auditing and Accounting Expert Systems," Decision Sciences 18, 1987, pp. 468-485.

Ohlsson, S. "Some Principles of Intelligent Tutoring," Instructional Science 14, 1986a, pp. 293-326.

Ohlsson, S. "Rational vs. Empirical Learning," In Information Processing, Ed. Kugler, H. J., Amsterdam: Elsevier Science Publishers, 1986b, pp. 841.

Ohlsson, S. and Langley, P. "Psychological Evaluation of Path-Hypothesis in Cognitive Diagnosis," In Learning Issues for Intelligent Tutoring Systems, Ed. Mandl, H. and Lesgold, A., New York, NY: Springer-Verlag, 1988, pp. 42-62.

Ohlsson, S. "Computer Simulation and its Impact on Educational Research and Practice," International Journal of Educational Research 12, 1988, pp. 534.

Ohlsson, S. and Rees, E. "The Function of Conceptual Understanding in the Learning of Arithmetic Procedures," Cognition and Instruction 8(2), 1991, pp. 103 - 179.

Paradice, D. B. and Courtney, J. F. "Organizational Knowledge Management," Information Resources Management Journal 2(3), 1989, pp. 1-13.

Partridge, D. A New Guide to Artificial Intelligence, Norwood, NJ: Albex Publishing Corporation, 1991.

Pau, L. F., Mottiwalla, J., and Pao, Y. H. (ed.) Expert Systems in Economics, Finance and Banking, Amsterdam: North Holland, 1989.

Payne, J. W. "Task Complexity and Contingent Processing in Decision-making: An Information Search and Protocol Analysis," Organizational Behavior and Human Performance 16, 1976, pp. 366-387.

Payne, J. W., Bettman, J. R., and Johnson, E. J. "Adaptive Strategy Selection in Decision-making," Journal of Experimental Psychology: Learning, Memory, and Cognition 14, 1988, pp. 534-552.

Payne, S. J. and Squibb, H. R. "Algebra Mal-Rules and Cognitive Account of Error," Cognitive Science 14, 1990, pp. 445-481.

Payne, J. W., Bettman, J. R., and Johnson, E. J. "The Adaptive Decision Maker: Effort and Accuracy in Choice," In Insights in Decision-making: A Tribute to Hillel J. Einhorn, Ed. Hogarth, R. M., Chicago, IL: The University of Chicago Press, 1990, pp. 129-153.

Pazzani, M. J. "Integrating Explanation-Based and Empirical Learning Methods in OCCAM," In Proceedings of the Third European Working Session on Learning, Ed. Sleeman, D., London, England: Pitman Publishing, 1988, pp. 147-166.

Pazzani, M. J. Creating a Memory of Causal Relationships: An integration of Empirical and Explanation-Based Learning Methods, Hillsdale, NJ: Lawrence Erlbaum Associates, 1990.

Pearl, J. Heuristics: Intelligent Search Strategies for Computer Problem Solving, Reading, MA: Addison-Wesley, 1984.

Peters, J. M., Lewis, B. L., and Dhar, V. "Assessing Inherent Risk During Audit Planning: The Development of a Knowledge Based Model," Accounting, Organizations and Society 14(4), 1989, pp. 359-378.

Piaget, J. The Child's Conception of the World, New York, NY: Harcourt, 1929.

Piaget, J. Success and Understanding, Cambridge, MA: Harvard University Press, 1976.

Polson, M. C. and Richardson, J. J. (ed.) Foundations of Intelligent Tutoring Systems, Hillsdale, NJ: Lawrence Erlbaum Associates, 1988.

Posner, M. J. (ed.) Foundations of Cognitive Science, M.I.T, 1989.

Post, E. "Formal Reductions of the General Combinatorial Problem," American Journal of Mathematics 65, 1943, pp. 197-268.

Psotka, J., Massey, L. D., and Mutter, S. A. (ed.) Intelligent Tutoring Systems: Lessons Learned, Hillsdale, NJ: Lawrence Erlbaum Associates, 1988.

Pylyshyn, Z. W. "The Role of Competence Theories in Cognitive Psychology," Journal of Psycholinguisitic Research 2(1), 1973, pp. 21-50.

Pylyshyn, Z. W. "Computation and Cognition: Issues in the Foundations of Cognitive Science," The Behavioral and Brain Sciences 3, 1980, pp. 11-169.

Pylyshyn, Z. W. Computation and Cognition: Toward a Foundation for Cognitive Science, Cambridge, MA: The MIT Press, 1984.

Pylyshyn, Z. W. "Computing in Cognitive Science," In Foundations of Cognitive Science, Ed. Posner, M. I., Cambridge, MA: MIT press, 1989, pp. 49-92.

Raiffa, H. Decision Analysis: Choice Under Uncertainty, Reading, MA: Addison Wesley, 1974.

Reason, J. Human Error, Cambridge, MA: Cambridge University Press, 1990.

Resnick, L. B. "Syntax and Semantics in Learning to Subtract," In Addition and Subtraction: A Cognitive Perspective, Ed. Carpenter, T. P. and Romberg, T. A., Hillsdale, NJ: Lawrence Erlbaum Associates, 1982, pp. 136-155.

Resnick, L. B. "A Developmental Theory of Number Understanding," In The Development of Mathematical Thinking, Ed. Ginsburg, H. P., New York, NY: Academic Press, 1983, pp. 109-151.

Resnick, L. B. "Cognition and Instruction: Recent Theories of Human Competence," In Psychology and Learning, Ed. Hammonds, B. L., Washington, D. C.: American Psychological Association, 1984, pp. 127-186.

Resnick, L. B. and Neches, R. "Factors Affecting Individual Differences in Learning Ability," In Advances in the Psychology of Human Intelligence, Ed. Sternberg, R. J., Hillsdale, NJ: Lawrence Erlbaum, 1984, pp. 275-323.

Resnick, L. B. "Constructing Knowledge in School," In Development and Learning: Conflict or Congruence ?, Ed. Liben, L. S., Hillsdale, NJ: Lawrence Erlbaum Associates, 1987, pp. 19-50.

Resnick, L. B. and Omanson, S. F. "Learning to Understand Arithmetic," In Advances in Instructional Psychology, Ed. Glaser, R., Hillsdale, NJ: Lawrence Erlbaum Associates, 1987, pp. 41-95.

Rich, E. Artificial Intelligence, New York, NY: McGraw-Hill Book Company, 1983.

Riesbeck, C. K. and Schank, R. C. Inside Case-Based Reasoning, Hillsdale, NJ: Lawrence Erlbaum Associates, 1989.

Riley, M. S., Greeno, J. G., and Heller, J. I. "Development of Children's Problem-Solving Ability in Arithmetic," In The Development of Mathematical Thinking, Ed. Ginsburg, H. P., New York, NY: Academic Press, 1983, pp. 153-196.

Rosenbloom, P. and Newell, A. "Learning by Chunking: A Production System Model of Practice," In Production System Models of Learning and Development, Ed. Klahr, D., Langley, P., and Neches, R., Cambridge, MA: The MIT Press, 1987, pp. 221-285.

Rosenbloom, P. S., Laird, J. E., Newell, A., and McCarl, R. "A Preliminary Analysis of the Soar Architecture as a Basis for General Intelligence," Artificial Intelligence 47, 1991, pp. 289-325.

Rumelhart, D. E. and Norman, D. A. "Analogical Processes in Learning," In Cognitive Skills and Their Acquisition, Ed. Anderson, J. R., Hillsdale, NJ: Lawrence Erlbaum Associates, 1981, pp. 335-360.

Rumelhart, D. E. and McClelland, J. L. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Cambridge, MA: The MIT Press, 1986.

Rumelhart, D. E., Smolensky, P., McClelland, J. L., and Hinton, G. E. "Schemata and Sequential Thought Processes in PDP Models," In Parallel Distributed Processing, Ed. McClelland, J. L., Rumelhart, D. E., and Group, P. R., Cambridge, MA: MIT Press, 1986, pp. 7-57.

Sauers, R. "Controlling Expert Systems," In Expert System Applications, Ed. Bolc, L. and Coombs, M., Berlin, Germany: Springer-Verlag, 1988, pp. 79-197.

Schank, R. C. and Abelson, R. P. Scripts, Plans, Goals, and Understanding: An Enquiry into Human Knowledge Structures, Hillsdale, NJ: Lawrence Erlbaum Associates, 1977.

Schoemaker, P. J. H. "The Quest for Optimality: A Positive Heuristic of Science?," Behavioral and Brain Sciences 14, 1991, pp. 205-245.

Self, J. "Student Models in Computer-Aided Instruction," International Journal of Man-Machine Studies 6, 1974, pp. 197-221.

Self, J. (ed.) Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction, New York, NY: Chapman and Hall, 1988.

Shanteau, J. "Cognitive Heuristics and Biases in Behavioral Auditing: Review, Comments and Observations," Accounting, Organizations and Society 14, 1989, pp. 165-177.

Shields, M. D. "An Analysis of Experimental Accounting Research on Managerial Decision-making," In Behavioral Accounting Research: A Critical Analysis, Ed. Ferris, K. R., Columbus, OH: Century VII Publishing Company, 1988, pp. 203-227.

Silver, E. A. "Using Conceptual and Procedural Knowledge: A Focus on Relationships," In Conceptual and Procedural Knowledge: The Case of Mathematics, Ed. Hiebert, J., Hillsdale, NJ: Lawrence Erlbaum Associates, 1986, pp. 181-197.

Simon, H. A. "A Behavioral Model of Rational Choice," Quarterly Journal of Economics 69, 1955, pp. 174-184.

Simon, H. A. The New Science of Management Decision, New York, NY: Harper and Row, 1960.

Simon, H. A. Administrative Behavior: A Study of Decision-Making Processes in Administrative Organization, New York, NY: The Free Press, 1976.

Simon, H., A. The Sciences of Artificial, Cambridge, MA: MIT press, 1979.

Simon, H. A. "Search and Reasoning in Problem Solving," Artificial Intelligence 21, 1983, pp. 7-19.

Simon, H. A. and Kaplan, C. A. "Foundations of Cognitive Science," In Foundations of Cognitive Science, Ed. Posner, M. I., Cambridge, MA: MIT press, 1989, pp. 1-47.

Simon, H. A. "How to Win at Twenty Questions with Nature," In Models of Thought, Ed. Simon, H. A., New Haven, CT: Yale University Press, 1989, pp. 20-29.

Simon, H. A. (ed.) Models of Thought, New Haven, CT: Yale University Press, 1989.

Simon, H. A. "Invariants of Human Behavior," Annual Review of Psychology 41, 1990, pp. 1-19.

Simon, H. A. and Lean, G. "Problem Solving and Rule Induction: A Unified View," In Readings in Machine Learning, Ed. Shavlik, J. W. and Dietterich, T. G., San Mateo, CA: Morgan Kaufmann Publishers, 1990, pp. 26-37.

Singley, M. K. and Anderson, J. R. The Transfer of Cognitive Skill, Cambridge, MA: Harvard University Press, 1989.

Slatter, P. E. Building Expert Systems: Cognitive Emulation, New York, NY: John Wiley & Sons, 1987.

Sleeman, D. H. and Smith, M. J. "Modeling Student's Problem Solving," Artificial Intelligence 16, 1981, pp. 171-188.

Sleeman, D. H. and Brown, J. S. (ed.) Intelligent Tutoring Systems, New York, NY: Academic Press, 1982.

Sleeman, D. "Assessing Aspects of Competence in Basic Algebra," In Intelligent Tutoring Systems, Ed. Sleeman, D. and Brown, J. S., New York, NY: Academic Press, 1982, pp. 185-199.

Sleeman, D., Langley, P., and Mitchell, T. M. "Learning from Solution Paths: An Approach to the Credit Assignment Problem," The AI Magazine (Spring), 1982, pp. 48-52.

Sleeman, D. "An Attempt to Understand Student's Understanding of Basic Algebra," Cognitive Science 6, 1984, pp. 387-412.

Sleeman, D. "Basic Algebra Revisited: A Study with 14-year-olds," International Journal of Man-Machine Studies 22, 1985, pp. 127-149.

Slovic, P., Fischhoff, B., and Lichtenstein, S. "Behavioral Decision Theory," Annual Review of Psychology 28, 1977, pp. 1-39.

Smith, D. A. and Greeno, J. G. "A Model of Competence for Counting," Cognitive Science 13, 1989, pp. 183-211.

Sternberg, S. "Two Operations in Character Recognition: Some Evidence from Reaction-Time Measurements," Perception and Psychophysics 2, 1967, pp. 45-53.

Sternberg, S. "Memory-Scanning: Mental Processes Revealed by Reaction Time Experiments," Acta Psychology 30, 1969, pp. 276-315.

Stevens, A., Collins, A., and Goldin, S. E. "Misconceptions in Students' Understanding," In Intelligent Tutoring Systems, Ed. Sleeman, D. and Brown, J. S., New York, NY: Academic Press, 1982, pp. 13-24.

Tchogovadze, G. G., Gogichaishvili, G. G., and Abbasov, I. I. "Principles of Intelligent Learning Systems Design," International Journal of Man-Machine Studies 28, 1988, pp. 391-416.

Tulving, E. Elements of Episodic Memory, New York, NY: Oxford University Press, 1983.

Tversky, A. and Kahneman, D. "Judgment under Uncertainty: Heuristics and Biases," Science 185, 1974, pp. 1124-1131.

Tversky, A. and Kahneman, D. "The Framing of Decisions and the Psychology of Choice," Science 211, 1981, pp. 453-458.

Ungson, G. R. and Braunstein, D. N. (ed.) Decision-making: An Interdisciplinary Inquiry, Boston, MA: Kent Publishing Company, 1982.

VanLehn, K. and Brown, J. S. "Planning Nets: A Representation for Formalizing Analogies and Semantic Models of Procedural Skills," In Cognitive Process Analyses of Learning and Problem Solving, Ed. Snow, R. E., Frederico, P. A., and Montague, W. E., Hillsdale, NJ: Lawrence Erlbaum Associates, 1980, pp. 95-137.

VanLehn, K. "Bugs are not Enough: Empirical Studies of Bugs, Impasses and Repairs in Procedural Skills," The Journal of Mathematical Behavior 3(3), 1982, pp. 3-71.

VanLehn, K. "On the Representation of Procedures in Repair Theory," In The Development of Mathematical Thinking, Ed. Ginsburg, H. P., New York, NY: Academic Press, 1983, pp. 197-252.

VanLehn, K., Brown, J. S., and Greeno, J. G. "Competitive Argumentation in Computational Theories of Cognition," In Methods and Tactics in Cognitive Science, Ed. Kintsch, W., Miller, J., and Polson, P., Hillsdale, NJ: Lawrence Erlbaum Associates, 1984, pp. 235-262.

VanLehn, K. "Arithmetic Procedures Are Induced from Examples," In Conceptual and Procedural Knowledge: The Case of Mathematics, Ed. Hiebert, J. H., Hillsdale, NJ: Lawrence Erlbaum Associates, Inc., 1986, pp. 133-179.

VanLehn, K. and Garlick, S. "Cirrus: An Automated Protocol Analysis Tool," In Proceedings of the Fourth Machine Learning Workshop, Ed. Langley, P. Los Altos, CA: Morgan Kaufmann, 1987, pp. 205-217.

VanLehn, K. "Learning One Subprocedure per Lesson," Artificial Intelligence 31, 1987, pp. 1-40.

VanLehn, K. "Student Modeling," In Foundations of Intelligent Tutoring Systems, Ed. Polson, M. C. and Richardson, J. J., Hillsdale, NY: Lawrence Erlbaum, 1988, pp. 55-77.

VanLehn, K. "Problem Solving and Cognitive Skill Acquisition," In Foundations of Cognitive Science, Ed. Posner, M. I., Cambridge, MA: The MIT Press, 1989, pp. 527-579.

VanLehn, K., Ball, W., and Kowalski, B. "Non-LIFO Execution of Cognitive Procedures," Cognitive Science 13, 1989, pp. 415-465.

VanLehn, K. Mind Bugs: The Origins of Procedural Misconception, Cambridge, MA: The MIT Press, 1990.

VanLehn, K. (ed.) Architectures for Intelligence, Hillsdale, NJ: Lawrence Erlbaum Associates, Inc., 1991.

VanLehn, K. "Rule Acquisition Events in the Discovery of Problem-Solving Strategies," Cognitive Science 15, 1991, pp. 1-47.

Vasarhelyi, M. (ed.) Artificial Intelligence in Accounting and Auditing Using Expert Systems, New York, NY: Markus Wiener Publishing, Inc., 1987.

Waller, W. S. and Felix, W. L. "Auditors' Covariation Judgments," Accounting Review , 1987, pp. 275-292.

Waterman, D. A. "Generalization Learning Techniques for Automating the Learning of Heuristics," Artificial Intelligence 1, 1970, pp. 121-170.

Waterman, D. A. "Adaptive Productions Systems," In Proceedings of the Fourth International Joint Conference on Artificial Intelligence, Tbilisi, USSR: 1975, pp. 296-303.

Waterman, D. A. and Hayes-Roth, F. (ed.) Pattern-Directed Systems, New York, NY: Academic Press, 1978.

Waterman, D. A. Building Expert Systems, New York, NY: McGraw-Hill Book Company, 1983.

Wenger, E. Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge, Los Altos, CA: Morgan Kaufmann Publishers, 1987.

Wilkins, D. C., Clancey, W. J., and Buchanan, B. G. "Using and Evaluating Differential Modeling in Intelligent Tutoring and Apprentice Learning Systems," In Intelligent Tutoring Systems: Lessons Learned, Ed. Psotka, J., Massey, L. D., and Mutter, S. A., Hillsdale, NJ: Lawrence Erlbaum Associates, 1988, pp. 257-277.

Winston. Artificial Intelligence, Reading, MA: Addision-Wesley Publishing Company, 1984.

Winston, P. H. and Horn, B. K. P. LISP, Reading, MA: Addison-Wesley Publishing Company, 1989.

Winterfeldt, D. V. and Edwards, W. Decision Analysis and Behavioral Research, New York, NY: Cambridge University Press, 1986.

Wright, G. (ed.) Behavioral Decision-making, New York, NY: Plenum Press, 1985.

Young, R. M. and O'Shea, T. "Errors in Children's Subtraction," Cognitive Science 5, 1981, pp. 153-177.